2002

A centralized object-relational database-based code services retrieval system tool for software reuse.

Xiaoquan. Zhao
University of Windsor

A Centralized Object-Relational Database Based Code Services Retrieval System Tool for Software Reuse

By

Xiaoquan Zhao

A Thesis

Submitted to the Faculty of Graduate Studies and Research
Through the School of Computer Science
In Partial Fulfillment of the Requirements for
The Degree of Master of Science at the
University of Windsor

Windsor, Ontario
Canada

Abstract

We describe a code services retrieval system tool designed and implemented within a grid test-bed environment. Among the problems and challenges for code services systems included are: how to store; and, how to query, access and build the code services efficiently within the context of distributed storage sites.

This thesis proposes a solution that integrates object-relational database technology, the Java JINI concept and JDBC technologies together to establish a centralized object-relational database based code services retrieval system tool for software reuse. In this thesis, we focus on how to store the code services and find which system architecture is suitable for code services retrieval system. We design and implement generic user-defined data types, including Character Large Object (CLOB) and Binary Large Object (BLOB) to store source codes and corresponding binary codes respectively. We use pure Java client-side JDBC to provide access to our new data types (CLOB and BLOB). The code services stored in the centralized object-relational database can be accessed remotely across networks. The portability of the centralized database is enhanced through use of Java. This system is access-effective.

Dedication

This paper is dedicated to:

My parents
My brothers
My teachers,
My wife

For their Love and Support!

Acknowledgements

I would like to acknowledge the support and advice of my supervisor, Dr. Robert D. Kent. My thesis work would have been impossible to finish without his giving me the opportunity to conduct my research throughout my entire Master study. He is both my mentor and my good friend.

I thank my thesis committee members, Dr. Phil A. Graniero and Dr. Alioune Ngom who give me a lot of valuable suggestions. I also thank Dr. Joan Morrissey who gave me valuable advices for my thesis proposal. I thank Dr. Akshai Aggarwal as my thesis defense chairman.

Special acknowledgement is expressed to SHARCnet (www.sharcnet.ca) for financial support during the thesis research.

I also want to thank my wife, Fang An for her understanding and help.

My thanks go to my good friends who have helped me so much during my Master's study at University of Windsor, Canada.

Table of Contents

Chapter 1: Introduction

1.1: Concepts of the Code Services Retrieval System:

A Code Service Retrieval System (CSRS) is a distributed system based on the idea of forming a group of collaborating users and the code services that are requested by those users. On one hand, the CSRS should enable remote user access to the code services in a straightforward manner and it should allow user to change their access locations. On the other hand, it should provide the appropriate functionalities to build, manage and maintain the code services as well as manage user transactions. The CSRS consists of a set of components that provides system infrastructure for code service in networked environment: example services include offering various language source codes, binary codes or combinations of the two. It should support collaborative programming and software reuse, the codes may be reused by their designers or by members of the collaborative group.

1.2: Software Reuse:

Our Code Services Retrieval System is software reuse system. Software reuse means "the process of using existing software components rather than building them from the scratch" [KRUE92]. There are two concepts underlying reuse [QIANG99]: first, one tries to avoid reinventing those software components which have already been invented; second, one tries to use already tested and proven software components as building blocks to construct new systems or modify existing systems.

Software reuse has also been recognized as an efficient way to improve both the productivity and the quality of new software projects [FRAK94, LIM94]. It increases software productivity because software reuse uses created, tested

and documented software components to build new system. It also improves the reliability because used software components are tested and proved in real application system, they are more reliable than those new components that are still tested in a simulated test environments. Ideally, components should be used without change to have confidence in their reliability. The reuse of software components that have been already tested may reduce the costs of software development. We all know, the cheapest component is one that does not require us to write, but just get it from other place. Moreover, reuse-oriented software development can reduce the maintenance cost, because maintenance operations on a modularized system can be better localized [ZHANG00, ZHONG00, LEO99].

Reuse may occur within a same software system, across similar software systems or widely different software systems. In this thesis, we focus on the code programs reuse mainly, our system provides Java, C++ and C source code reuse and corresponding binary code reuse. The Java, C++ and C codes could be reused by the code consuming users who want to use them in their programs. The code providing users could insert their own useful Java, C++ and C codes into our system.

1. 3: Motivation of the Thesis:

Currently, the problems and challenges for Code Service Retrieval System are:

- How to store the code services.
- How to query/search the code services.
- Which system architecture is suitable for the code service retrieval system.
- How to build and classify the code services.
- What kind of system architecture is cost effective?

- System security problem.
- System reliability problem.
- How will the system support the collaborative programming and software reuse.

From all of these, we think that the most important and challenging problems and issues are: How to store the code service and find which system architecture is suitable for code services retrieval system. Previous and existing systems are: using traditional relational database to store the paths or locations of code services [RAMAK00], using distributed file systems and directory services to store code services and using eXtensible Markup Language (XML) to store code services. All these systems have weak code service storages and their system architectures are not suitable for software reuse and collaborative programming under distributed networked environment. And also their system costs are not effective.

Thus, how to solve the problem of storage of code services and find which system architecture is suitable for code services retrieval system to support software reuse are the motivation of this thesis.

1. 4: Objective of the Thesis:

The objective of this thesis is to design and implement a centralized object-relational database based code service retrieval system tool for Collaborative Programming and Software Reuse. We mainly focus on the storage of the code service and system architecture that is a centralized object-relational database based code service retrieval system architecture.

The goals of our thesis project are:

- Propose and implement a new approach to store the code services by creating our own new user-defined data types including Character Large Object (CLOB), Binary Large Object (BLOB) to store source codes, corresponding binary codes by using Object-Relational Database.
- Design and construct a centralized object-relational database based architecture by using new pure Java client-side JDBC to access new data types including CLOB and BLOB from remote location by users, it is more cost-effective and more suitable for current code services system.
- Implement the basic functionalities: how to query/search and classify the code service because they are necessary and essential components for code service system.

1.5: What is the Exact Meaning of "Code Services"?

Code Services means:

1. **Code Provider Service:** This service allows user to insert, build, update, and maintain the computer language source codes and corresponding binary codes, files from remote locations.
2. **Code Consumer Service:** This service allows user to query, access and retrieve computer language source codes and corresponding binary codes, files from remote locations.
3. All the service transaction operations are processed on a distributed networked environment.
4. Our system just provides the source codes and corresponding binary codes' services for Java, C++ and C languages.

1.6: Organization of the Thesis:

This thesis is organized into 5 chapters:

Chapter 1 describes the general concepts of the code service retrieval system first, then gives the simple introduction of software reuse, and then gives the motivation for this thesis, objectives of the thesis and the exact meanings of " Code Services " in my thesis. Finally, organization of this thesis is addressed.

Chapter 2 discusses the object-relational database based code storage and retrieval, we explain why we need create my own user-defined data types including character large object (CLOB) data type and binary large object (BLOB) data type to store source codes and corresponding binary codes, and then explain what CLOB and CLOB and their storages are. We discuss how to retrieve these data types.

Chapter 3 discusses the various architectures of code service systems such as CORBA, RMI, JINI, Socket, JDBC first, and then analyze and compare their system costs. Finally, we give our solution for code service system.

Chapter 4 discusses our system prototype design and implementation in detail, we discuss our system architecture and system costs first, and then discuss the system design specification and system implementation. And finally, we discuss how to establish our system and how to use our system's code provider component and code consumer component in detail.

Chapter 5 discusses the system testing first, and then gives out our thesis conclusion, followed by some ideas for the future work.

Chapter 2: Object-Relational Database Based Code Storage and Retrieval

2.1 Why Do We Need Object-Relational Database?

It is well known that traditional relational database systems just support a small, fixed collection of data types such as integer, date, string, that have proven adequate for traditional application domains such as administrative data processing. The most important feature of relational database systems, however, is not their ability to store and retrieve data, but their ability to search the data that they manage. It is primarily their flexible search capabilities that set relational database systems apart from first-generation database systems and file systems. Relational database systems provide a powerful, high-level query language (SQL) for use in searching. They also provide views and access control to enable the needs of different groups of users to be individually met [RAMAK00, IBM00, KELLER01, MACHA96].

But, in many application domains, much more complex kinds of data such as Large Object data must be handled. Typically these complex data have been stored in OS file systems or specialized data structures, rather than in a DBMS. Examples of domains with complex data include computer source code, executable binary code, multimedia repositories, and document management.

As the amount of data grows, the many features offered by a DBMS, for example, reducing application development time, concurrency control and recovery, indexing support, and query capabilities become increasingly attractive and ultimately necessary. In order to support such applications, a DBMS must support complex data types such as Large Object Data Type. Object-oriented concepts have strongly influenced efforts to enhance database support for complex data and have led to the development of object-relational

database systems.

Moreover, in today's world, enterprises need to access these Large Objects and relate them to their existing numeric and character data. To manage and exploit these new kinds of data effectively, today's enterprises need a database system that supports integrated content search, they need the ability to issue SQL queries that can simultaneously filter and relate both their traditional SQL data as well as the new, non-traditional, more complex types of data objects. This need is pushing relational database systems to evolve into object-relational [STON96] database system, which will be used in our system's design.

Our system will use object-relational database, we can create our own new user-defined data types including Large Objects such as Binary Large Object (BLOB) and Character Large Object (CLOB) to store our source codes and corresponding binary codes.

2.2: Previous and Existing System Design for Code Storage and Retrieval:

The following is a summary of previous and existing systems for code storage and retrieval.

- Some systems use traditional relational database to store the paths or locations, the actual source codes are stored outside of database as file format, these systems can not use the major functionalities of database such as data integrity, data transaction, data synchronization and data recovery.
- Some systems use distributed file systems and directory services to store source codes as file format. The operating system's file system manages

these files, they do not support query and transaction for these codes.

- Software Library Approaches by former students Zhong, Sheng and Zhang, Hui(M.Sc. Theses, 2000), just offer text source code storage and do not support corresponding binary code storage such as executable binary code, image files, audio files and video files[ZHONG00, ZHANG00].
- Other systems use eXtensible Markup Language (XML) to store source codes, but they do not support to store corresponding binary codes. Moreover, these systems are too complex because one must define different tags and elements for different codes.

All the above systems just store and handle the simple source codes, they do not support the corresponding executable binary codes, byte-codes, text files and image, audio and video binary files. These codes are needed when you want to run the program on the run time. So, our code service system will propose an approach to store all these codes and files, and also to manipulate them.

2.3 The Object-Relational Model:

The object-relational model is an evolutionary way to introduce object-oriented features to the traditional database without giving up the existing relational database features that are used in existing applications. In other words, object-relational database can be thought of as an attempt to extend relational database systems with the functionality necessary to support a broader class of applications, and in many ways, provide a bridge between the relational and object-oriented paradigms [RAMAK00, XHUMA97].

2. 4: User-Defined Data Types and Object Types:

Traditional database just provides the built-in data types such as char,

varchar2, number and date. Each column value and constant in a SQL statement has a built-in data type, which is associated with a specific storage format, constraints, and a valid range of values. When we create a table, we must specify a built-in data type for each of its columns. But these built-in data types are not enough for software applications. Object-relational database allows users to define their own data types according to user's need. These kinds of data types are called "User-Defined Data Type", they are additional kinds of data types that specify both the structure of the data and the ways of operating on it, and these types are used within the relational model. On this way, this approach adds value to the data stored in a database. User-defined data types make it easier for application developers to work with complex data like large objects such as source codes, binary codes, image files, audio files, and video files. Actually, user-defined data types are object types that store structured business data in its natural form and allow applications to retrieve them. For this reason, they work efficiently with applications developed using object-oriented programming techniques [RAMAK00, ORACLE99].

User-defined data types are object types that are abstraction of the real-world entities, they are analogous to Java and C++ classes. One can think of a user-defined data type as an object template, and an object as a structure that matches the template. Object types can represent many different data structures. One can use object type to map an object model directly to a database schema, instead of flattening the model to relational tables and columns. It thus brings related pieces of data together to a single unit, and to store the behavior of data along with the data itself. Application code can retrieve and manipulate the data as an object.

A user-defined data type (object type) is a schema object with three kinds of components:

- **A name**, which identifies the object type uniquely within that schema.
- **Attributes**, which model the structure and state of the real-world entity. Attributes can be built-in types or other object types.
- **Methods**, functions or procedures that implement operations that mimic ones we can perform on the real-world entities.

When you create a variable of a user-defined data type (object type), the result is an object. The object has attributes and methods based on its type. Because the object is a concrete thing, you can assign values to its attributes and call its methods.

2.5: Storage of User-Defined Types and Object Types:

Object-relational database stores and manages data of user-defined types and object types in tables. It can automatically map the complex structure of user-defined types and object types into the simple rectangular structure of tables [ORACLE99].

- **Leaf-Level Attributes:**

The structure of an object type is very similar to a tree format. The branches that grow from the trunk go to the attributes. If an attribute is of a user-defined type or object type, that branch sprouts sub-branches for the attributes of the new user-defined type or object type.

Here, in order to understand the structure of an object type easy, we use the tree and trunk terms, it is different from the tree term used by data structure class such as general tree and binary tree. The tree in data structure class consists of a set of nodes and a set of edges. The structure of an object type

has a main trunk, and branches that grow from the trunk go to the attributes.

Ultimately, each branch comes to an end at an attribute that is of a built-in type or a collection type. These are called leaf-level attributes of the original object type. Object-relational database provides a table column for each leaf-level attribute.

The leaf-level attributes that are not collection types are called the leaf-level scalar attributes of the user-defined type or object type.

- **Row Objects**

In an object table (Object table is the table which is defined by using user-defined data types or object types), every leaf-level scalar or reference attribute has a column in which object-relational database stores its actual data. Object-relational database stores leaf-level attributes of table types in separate tables associated with the object table. You must declare these tables as part of the object table declaration.

Access to individual attributes of objects in an object table is simply access to columns of the table. Accessing the value of the object itself causes object-relational database to invoke the default constructor for the type, using the columns of the object table as arguments. It stores the system-generated object identifier in a hidden column. It uses the object identifier to construct references to the object.

- **Column Objects:**

When a table is defined with a column of a user defined data type or object type, object-relational database adds hidden columns to the table for the

object type's leaf-level attributes. An additional column stores the NULL information of the object (that is, the atomic nulls of the top-level and the nested objects).

- **References (REF)**

Object-relational database constructs a reference to a row object by invoking the built-in function REF on the row object. The constructed REF is made up of the object identifier, some metadata of the object table.

The size of a REF in a column of REF type depends on the storage properties associated with that column.

Object-relational database stores unconstrained REF values in a single column. If a REF column is scoped or referentially constrained to an object table with the system generated object identification (OID), object-relational database creates a single column to store the OID value. However, if the OID is primary-key based, then object-relational database may create one or more internal columns to store the values of the primary key depending on how many columns comprise the primary key. References to row objects containing primary key based on OID may only be stored in REF columns with scope or referential constraints.

2.6: An Approach to Store the Code (Storage of Code)

From the discussion above, we know the approaches for storages of previous and existing systems are weak and limitary. Therefore, we introduce the object-relational database. We think that it is good approach to store source code and corresponding binary code by creating our own User-defined Data Type, we can create new User-defined data type including Large Object data

types such as Character Large Object (CLOB) data type and Binary Large Object (BLOB) data type to store our source code and corresponding binary code. We use Character Large Object (CLOB) data type to store source codes such as Java, C++ and C, and use Binary Large Object (BLOB) data type to store corresponding binary codes such as executable binary codes, image, audio and video files. With the development of software, many software today include related image, audio and video data, so, our system should store these data.

User-defined data type is an object type. Once the source code and corresponding binary code are stored into the user-defined data type, the instance of this user-defined data type is an object. We can use various object-oriented programming functionalities to handle it, it is more easy and efficient.

2.6.1: The Structure of Code:

Before we define our own user-defined data type to store the codes, we must analyze the structure the code.

A complete executable program code of Java, C++ and C should have three kinds of files, we call them: Main Code, Dependent Code and Binary Code.

1: Main code:

Every Java, C++ and C program code must have a method or function called main (), the program starts by executing this main () method or main () function; For Java and C++, we call the class which contains the main () method or function as main code; similarly for C program code, we call the file which contains the main () function as main code. Main code is character

file. From the view of storage point, the structure of main code in our system is defined as:

- **CODEID** (Code Identification) each code has a unique code ID, and should be the number data type.
- **USERID** (User Identification) each user who uses the code or provide the code must have a unique user ID. But we do not provide the user group, there is no group ID.
- **FILENAME** (File Name) file name is the initial full path or location created by code owner in his or her computer.
- **CODENAME** (Code Name) each code has a code name that represents the main function of this code.
- **CODETYPE** (Code Language Type) Our system just provides the Java, C++ and C code services, so only has three code language types.
- **COMPILERTYPE** (Compiler Type) which is the compiler that compiles the source code into corresponding executable binary code or byte-codes (only for Java).
- **PLATFORM** (Executable Platform) is the platform that runs corresponding executable binary code.
- **AUTHOR** (Author of the Code) is the author that creates the source code and corresponding binary code.
- **VERSION** (Version Number) is the version number of the code.
- **CODEDESCRIPTION** (Description of the Code) it is the description of the functions of the code in detail.
- **CODE** (Actual Code) store actual source code of main code, we use Character Large Object (CLOB) data type to store source codes such as Java, C++ and C. It is one of the key parts of this thesis project.

2: Dependent Codes:

Dependent codes are the codes on which the main code depends during compiling time and run-time. Main code must use and work together with the dependent codes to finish the work. The main code calls dependent codes. In other words, the main code uses them as arguments in the signature of operations. Dependent codes are also character files.

We do not consider the standard library such as .dll and .so as the dependent codes because standard platform uses same library during compile time and run time. If user uses his/her own library rather than the standard library, he/she should add his/her own library into the dependent codes.

- **CODEID** (Code Identification) is same as the code ID of main code defined above, and also should be the number data type.
- **FILENAME** (File Name) file name is the initial full path or location created by code owner in his or her computer for corresponding dependent code.
- **CODE** (Actual Code) very similarly to the above code of main code, it also stores the actual source code of dependent code, we also use Character Large Object (CLOB) data type to store corresponding dependent codes such as Java, C++ and C. It is also one of the key parts of this thesis project.

3: Binary Codes:

There are two kinds of binary codes, one kind of the binary codes is the executable binary codes (for Java, it is byte-codes) that are created by the specified compiler; The other kind of binary code is the binary files that main code must depend on to run correctly on the run time or compiling time such as image binary files, audio binary file and video binary files, they all are binary format.

- **CODEID** (Code Identification) is same as the code ID of main code defined above, and also should be the number data type.
- **FILENAME** (File Name) file name is the initial full path or location created by code owner in his or her computer for corresponding binary codes.
- **CODE** (Actual Code) it stores actual binary code, we also use Binary Large Object (BLOB) data type to store corresponding binary codes of source code. It is also on of the key parts of this thesis project.

In order to meet the code integrity, all change operations for main code, dependent code and binary code are treated as one transaction. This transaction is either committed or roll backed.

2.7: Large Object Data Types (CLOB and BLOB)

As we mentioned above that we will store the codes (main codes, corresponding dependent codes and binary codes) into the Large Object data type such as the Character Large Object data type (CLOB) and Binary Large Object data type(BLOB). Therefore, in this section, we will discuss the CLOB and BLOB in detail.

2.7.1: Why Our System Need LOB (CLOB and BLOB)?

As applications development, we need to handle various kinds of data, for example, in our code service system, we need to store and deal with the various codes such as source codes, executable binary codes, byte-codes, text files, image binary files, even audio and video binary files, they are not simple structured data, actually, they are complex structured data and unstructured data, usually they are large. Traditionally, the relational database

was very successful in dealing with simple structured data that can be fit into simple tables, but, it cannot deal with the complex structured data and unstructured data. LOB data types (BLOB and CLOB) are designed to support the complex unstructured data and are optimized for large amounts of data storage. Because our code service system need to store the various files, so we use two variations of LOB data types such as BOLB, CLOB to store the codes and other files that are the unstructured data.

2.7.2: The Large Object(LOB) Data type:

Depending on the location of storage, LOBs are divided into two kinds: internal LOBs and external LOBs, external LOBs are also called BFILEs (binary files).
In the following, we will give the explanations for both in detail.

- **Internal LOBs:**

Internal LOBs, like their name suggests, are stored inside object relational database's table spaces in a way which optimizes data space and provides efficient data access. Internal LOBs support the transaction of database, for example, in the case of transaction or media failure of database, and any changes for an internal LOB value can be committed or rolled back. There are two kinds of data types for defining instances of internal LOB.

1. Binary Large object (BLOB), which is a kind of LOB whose value is composed of unstructured binary data.
2. Character Large object (CLOB), which is a kind of LOB whose value is composed of character data that corresponds to the database character.

- **External LOBs (BFILEs)**

External LOBs (BFILES) are large binary data objects that are not stored inside the object-relational database, they are stored inside the operating system's file system, and actually outside database table spaces. These files use reference semantics. They can be stored in conventional secondary storage devices such as hard disk, BFILEs can also be stored on tertiary block storage devices such as CD-ROMs, Photo CDs and DVDs. The BFILE data type allows Read-Only byte stream access to large files on the file system of the object relational database server. The Object-relational database server can access BFILEs provided the underlying server operating system supports stream-mode access to these operating system (OS) files. Binary files (BFILE) is the only one data type of external LOB Data types, whose value is composed of binary data, and is stored outside the database table spaces in a object relational database side operating system file.

Our code service system stores the codes (main code, corresponding dependent codes and binary codes) into internal LOBs such as the Character Large Object data type(CLOB) and Binary Large Object data type(BLOB) because the internal LOBs store the CLOB and BLOB inside the object-relational database, the database supports the data transaction. External LOBs such as BFILEs do not participate in database's transactions. Any support for data integrity and durability must be provided by the underlying file system as governed by the object relational database's operating system. So, external LOB data type is not suitable for our code service's storage.

2.8 Creating Own User-Defined Data Types Containing LOB Data Types

We can define our own user-defined data type that contains CLOB and BLOB to store the source codes and binary codes, and then define a table by using

our own user-defined data type, for example:

Firstly, we create the user-defined data type, we define this data type as an object type that contains the LOB attributes before we can proceed to create a table that makes use of that object type.

(I) Create a user-defined data type (Object Type) MY_CODE_TYPE that contains LOBs such as CLOB, BLOB. It is an object.

```
CREATE TYPE MY_CODE_TYPE AS OBJECT
    (CODE_ID                NUMBER(5),
     USER_ID                NUMBER(4),
     FILE_NAME              VARCHAR2(50),
     CODE_NAME              VARCHAR2(50),
     CODE_TYPE              VARCHAR2(10),
     COMPILER_TYPE          VARCHAR2(10),
     PLATFORM               VARCHAR2(20),
     AUTHOR                 VARCHAR2(30),
     VERSION                NUMBER(2),
     CODE_DESCRIPTION       CLOB,
     SOURCE_CODE            CLOB,
     EXECTUABLE_CODE        BLOB
    );
```

In the above, we defined our own user-defined data type: MY_CODE_TYPE, we define it as an object, it contains the CLOB, BLOB data types. Then we create a MY_CODE_TABLE by using the MY_CODE_TYPE just defined above. This table contains two kinds of LOB data types -- two CLOB and a BLOB.

Please note this approach inherits method set from the object-relational

database. User-defined methods are not included in our approach.

(II) Create table MY_CODE_TABLE by using our own defined data type:

CREATE TABLE MY_CODE_TABLE **OF** MY_CODE_TYPE;

This table is totally different from the table of traditional database, it is a table of object-Relational database, and also is an object table. So, in the object table of MY_CODE_TABLE, each column of the table maps to the each column of the MY_CODE_TYPE of our own defined data type.

The advantages of defining and using Large Object (LOBs) such as CLOB and BLOB to store our codes are:

a) LOBs are designed to support the large unstructured data, we all know that source codes and corresponding binary code of our codes service system are unstructured data and large.
b) All information of a source code and corresponding binary code are stored as a whole object. So, we can write and use functions (such as: getClob(), getBlob(), read(), substr(), insert(), write(), append(), empty(), copy(), trim() and so on) to manipulate the object, so, in this way, it is easy to insert, delete, update the object. In other words, it is easy to do many operations for a code (object).
c) LOBs are stored inside database table spaces rather than file system, which optimizes storage space and provides efficient data access. So, it allows user to access them much faster than traditional data types. Object-relational database can manage these database table spaces, all these database table spaces are parts of object-relational database. Object-relational database can guarantee these table spaces' transaction, integrity.

d) Because all codes (objects) are managed by object-relational database, so, the database can guarantee the following operation's processing of source code (object).

- **Code's Integrity:** the correctness of source code(object)when you insert, update, delete a code(object).
- **Code Transaction:** when you perform an atomic or consistent sequences of codes (objects) actions such as writing, reading, modifying. You can recover LOBs in the event of transaction or media failure, and any changes to a LOB value can be committed or rolled back.
- **Synchronizations (Concurrency control)** allows multi-users to access the codes (objects) at the same time.
- **Recovery:** survival of code crashes in any case.

2.9: Code Retrieval:

In the above, we have discussed the storage of codes, we use LOBs (CLOB and BLOB) to store the source codes and corresponding binary codes. In this section, we will discuss how to retrieve these codes that are stored inside the CLOB and BLOB.

2.9.1: Locators of Internal LOBs (BLOB and CLOB)

Actual data of CLOB or BLOB data types stored in a LOB is called the LOB's value. Depending on the size of internal LOB, the value of an internal CLOB or BLOB may or may not be stored inline with the other row data. If the internal LOB value is less than approximately 4000 bytes, then the value is stored inline; otherwise it is stored outside the row, but still inside the internal LOBs. Regardless of where the value of the internal LOB is stored, a locator

is stored in the row, and the object-Relational database can guarantee and manage data integrity, data transaction, data synchronizations (concurrency access control) and data recovery. You can think of a LOB locator as a pointer to the actual location of the LOB value. For internal LOBs, the LOB column stores a locator to the LOB's value that is stored in an object relational database's table space. Each LOB column/attribute for a given row has its own distinct LOB locator and copy of the LOB value stored in the database table space [ORACLE00].

2.9.2: Accessing the Internal LOBs (BLOB and CLOB)

Before you can start inserting data to an internal LOB, the LOB column/attribute must be made non-null. In other words, you must set a locator for this LOB. We usually can do this by initializing the internal LOB to empty in an INSERT/UPDATE statement of SQL using the functions EMPTY_BLOB() for BLOBs or EMPTY_CLOB() for CLOBs.

SELECT statement of SQL on a LOB usually just returns the locator instead of the actual value of LOB. Once you get the locator of the a LOB, you can do any and operations for this LOB, locator is mapped to locator pointer which is used to manipulate the LOB value. In other words, you must get the locator of the LOB before you want to use it.

Now let's have a look at complete transaction step for CLOB or BLOB by using the locator.

1. Get the locator of CLOB or BLOB you want to access.
2. Begin the transaction such as inserting, writing or deleting.
3. Use the locator to read, insert or delete actual data value from the CLOB or BLOB.

4. Commit when transaction is finish successfully, or rollback the transaction when you fail to finish it.

2.9.3: Using Java (JDBC) to Retrieve LOBs(CLOB, BLOB)

Java Database Connectivity (JDBC) is very good tool to write Java application to access the LOBs (CLOB and BLOB), actually the JDBC is a set of APIs that allow Java programs to connect to a database[ORACLE01]. Our system will use JDBC to access the LOBs, we can make changes to an entire internal LOB, or to pieces of the beginning, middle or end of an internal LOB in Java by means of the JDBC APIs. The JDBC interface will let you access both internal and external LOBs for read purposes, and you can also write to internal LOBs.

The BLOB and CLOB classes in JDBC provide methods for performing operations on large objects in the database including BLOB and CLOB data types. These classes (BLOB, CLOB) encapsulate LOB locators, so the user does not deal with locators but instead uses the methods and properties provided to perform operations and get state information. Any of LOB functionality not provided by these classes can be accessed by a call to the DBMS_LOB PL/SQL package.

We can get a reference to any of the above LOBs either as a column of an ResultSet or as an "OUT" type PL/SQL parameter from an PreparedStatement. When BLOB and CLOB objects are retrieved as a part of a ResultSet, these objects represent LOB locators of the currently selected row. If the current row changes due to a move operation (for example, rset.next()), the retrieved locator still refers to the original LOB row. In order to retrieve the locator for the most current row, you must call getXXXX() on the ResultSet each time when a move operation is made (where XXXX is a BLOB or CLOB).

2.10:Code's Classification:

Before the codes are stored into our centralized database, they should be classified based on classification and retrieval methods such as Descriptive classification and Type-Based retrieval. Descriptive classification is a classification method that specifies some attributes for the code, mostly focusing on the functional description of the code. Because of the relative easy-to-use of its concepts and the convenience to operate, this method is used often. According to the description of the code, we specify some attributes to the code such as Code Name, Code Feature, Language Type, Compiler Type, Executable Platform and Name of Author. These attributes are character string or text. For character string, we can search them. But for text, we cannot search it in our system, it cab be offered in the future work. We can also use Type-Based retrieval method to classify the different types of codes such as Java, C++ and C, so that users can query them by keywords easily. The reason to use classification is mainly to classify the unstructured codes into structured data so that they are easy to put into the object-relational database to allow user to browse and query them quickly.

Chapter 3: Architectures of Code Service Systems

In the section, we will discuss the general architectures for code service systems and their database system access cost performance. We will also discuss some system architectures that are used by previous and existing code systems, and then point out limitations of these systems. In the final, we will give the project system architecture of my thesis: a centralized object-relational database based code service architecture.

The architectures of most of the previous and existing systems primarily focused on either various server side scripting mechanisms to support database connectivity [HELM97, LAMB00], or evaluated the Java client/server communication paradigm [ORFA98]. Depending on the way the client establishes connection with the middleware, the architectures can be classified as two kinds: one is called Remote Procedure Call (RPC), this one has clear remote method invocation semantics, the concept RPC is simple - to make what appears to be a normal procedure call from within a process and has its execution actually carried out within some other process, usually on a remote system. Various implementations of RPC protocols have been developed with the common goal of reducing the complexity of communicating between processes through implementation hiding. The RPC client process makes a call to what appears to be a standard function, known as a stub. However, rather than executing locally, the parameters passed to the function are packaged and transmitted to a remote execution environment, where they are passed to the real implementation of the function. Upon completion of the function's execution, its return value is serialized and passed back to the client stub, which returns it to the caller [QUOIN00, VARAD99, DAVID00]. The other one is called Non Remote Procedure Call (Non-RPC), this does not provide for remote method invocation mechanism.

Common Object Request Broker Architecture (CORBA),Java Remote Method Invocation (RMI), Java JINI and Java JDBC are RPC; Sockets(including Java socket and C/C++ socket) and Java Servlets are Non-RPC.

3. 1: Common Object Request Broker Architecture (CORBA)

CORBA is powerful system architecture for heterogeneous networked systems. It has become a standard, and is managed by the Object Management Group (OMG) for the distribution of objects. It is designed as a platform-neutral infrastructure for inter-object communication, and has gained many big computer companies' acceptance.

CORBA defines client/server relationships between objects in a common interface language [SIEGEL96]. Unlike RMI that only implemented in Java, CORBA objects can be implemented in any programming language. So, it is very suitable for heterogeneous system that consists of various language implementations. It is concerned with the communication between objects implemented in different languages, using different platforms across the network. Java can be used for the implementation of these objects [BOUG00].

In order for a CORBA client object to utilize a CORBA server object, an implementation of CORBA's basic functionality called the Object Request Broker (ORB), has to be loaded at both the client and the server sites. The ORB offers the base architecture [CORBA97] as well as a number of services [COSS97] such as security, transactions and message.

CORBA allows applications to use the common interfaces. The interfaces are defined by Interface Definition Language (IDL). IDL supports multiple platforms and development tools. It is designed to be platform and language-independent [BAKER97]. Data and call format conversions are

handled transparently by the ORB. All interfaces to CORBA objects, and the data types used in those interfaces, are specified in IDL. This common definition allows applications to operate on objects without concern for the manner in which the object is implemented [QUOIN00].

Our code service system focuses on the database access. Let us take a look the cost for database access of CORBA [PAPA00]. The cost includes two phases: Initialization phase and execution phase, initialization phase is the procedure for establishing database connectivity, and execution phase is the procedure for querying the database after database connection is established.
The system cost of the database access for CORBA is:

A. Initialization phase:

1. The time for the client to initialize core ORB classes.
2. The time for the client to bind to the application server.

B. Execution phase:

1. The time for the client to invoke a method on the application server passing the SQL statement as a parameter.
2. The time for the application server to execute the SQL, obtain and return the results to the client.

The limitation of CORBA is: CORBA need to define static IDL information such as client side stub and server side skeleton in advance, it is static. But now, most distributed computing systems are dynamic, one can not predetermine what will be happened, new services might need to be added according to practical real time requirement, or some existing services could stop or crash. Also, it is very difficult to change the services because changing the stub or skeleton files requires to change all of the corresponding stub or skeleton files.

3.2: Java Remote Method Invocation (RMI)

Java Remote Method Invocation (RMI), that is relatively new to the computing industry, has gained a great deal of acceptance. Java RMI [RMI97] provides a language-specific architecture that allows Java-to-Java pure distributed applications to be built easily. RMI is a Java application interface for implementing remote procedure calls between distributed Java objects [DOWN98].

RMI consists of two separate parts: one is a server, and the other is a client. A typical server creates some remote objects, makes references to them accessible, and waits for clients to invoke methods on these remote objects. In other words, the server is responsible for handling requests by allowing clients to remotely invoke methods on it. A typical client gets a remote reference to one or more remote objects in the server and then invokes methods on them. RMI provides the mechanism by which the server and the client communicate and pass information back and forth [PAPA00].

The system cost of the database access for RMI is:

A. Initialization phase:

1. The time for the client calls a bind method of RMI to obtain a reference to the remote application server (binds to it).

B. Execution phase:

1. The time for the client to invoke a method on the application server passing the SQL statement as a parameter.
2. The time for the application server to execute the SQL statement, and then return the results.

The limitation of RMI is: It just supports Java-to-Java pure distributed application system, it dos not support the heterogeneous application systems.

3.3: Sockets and Java Servlets:

- Sockets usually consist of two parts: a client socket and a server socket, they both often come as pair. Socket is one endpoint of a two-way communication link between two programs running on the network. A socket is bound to a port number so that the TCP and UDP layer can identify the application that data will be sent. Socket is lower-level network communication. For example, when we want to write a client-server application, the server provides some service, such as processing database queries or sending out current stock prices. The client uses the service provided by the server, either displaying database query results to the user or making stock purchase recommendations to an investor. The communication that occurs between the client and the server is reliable if TCP is used, that is, no data can be dropped and it must arrive on the client side in the same order in which the server sent it. Socket use the data stream to access the data, the stream may be character stream or binary stream [PAPA00].

The system cost of database access for socket is:

A. Initialization phase:

1. The time for the client to open a socket connection with the application server.

B. Execution phase:

1. The time for the client to pass to the application server the data stream containing the SQL statement.
2. The time for the application server to execute the query, obtain the results, and return them to the client.

The limitation of socket is: Socket is lower-level network communication

programming, one must write your own code to handle the incoming and outcoming data streams, the system will be too complex if you establish the system which depends on the socket totally.

- **Java Servlets:**

Java Servlets technology today is primarily designed for use with the HTTP protocol of the World Wide Web, but servlets are being developed for other technologies. Servlets are effective for developing Web-based solutions that help provide secure access to a Web site, that interact with databases on behalf of a client, that dynamically generate custom HTML documents to be displayed by browsers and that maintain unique session information for each client.

Also many developers feel that servlets are the right solution for database-intensive applications that communicate with so-called thin clients-applications that require minimal client-side support. The server is responsible for the database access. Clients connect to the server using standard protocols available on all client platforms. Thus, the business logic code can be written once and reside on the server for access by clients [PAPA00].

The system cost of database access for Java servlets is:

A. Initialization phase:

1. The time for the client to open a URL connection with the Web server.

B. Execution phase:

1. The time for the applet to invoke, through the Web server, the corresponding servlet passing the SQL statement as a parameter (stating explicitly the servlet name and type of operation).
2. The time for the servlet to execute the request, obtain and return the

entire result table to the client.

The limitation of servlets is: servlets usually depend on middleware web server such as Weblogic, JRun and Apache, so, it needs the web server's support, it runs as a child process within the context of a web server.

3.4: Java JINI:

A Java Jini system is a distributed networked system which is based on the idea of federating groups of users and the resources required by those users [SUN00]. The overall goal is to turn the network into a flexible, easily administered tool on that resources can be found by human and computational clients. Resources can be implemented as either data resources, hardware devices, software programs, or a combination of them. The focus of the system is to make the network a more dynamic entity that better reflects the dynamic nature of the workgroup by enabling the ability to add and delete services flexibly.

Java Jini try to address the problems of distributed computing using a set of simple interfaces and protocols, Jini enables spontaneous networks of software services to assemble into working groups of objects, or federations. Jini enables self-healing when one or more services are removed from the federation, Jini builds on the Java2 Platform. Jini is usable for software resources as well as hardware services. Jini treats every thing as an object, everything for Jini is represented by Java Objects. Moreover, everything for Jini is located and accessed through Java interfaces [DAY00].

The most important concept within the Jini architecture is that of the service. A service is an entity that can be used by people, programs, or even other services. Jini systems provide mechanisms for these services' construction,

lookup, communication, and usage in a distributed networking environment. Services in a Jini system communicate with each other by using a service protocol, which is a set of interfaces written in the Java programming language.

Services are found and resolved by a lookup service. The lookup service is the central bootstrapping mechanism for the system and provides the major point of contact between the system and users of the system. In addition, descriptive entries associated with a service allow more fine-grained selection of services based on properties understandable to people.

The limitation of JINI is: It just supports to store small amount of services, not for large amount of services. It just supports LAN; does not support large scalable distributed computing. Good performance for supporting hardware devices services (small services, below 100, small storage); not good for large services storage.

3.5: Java Database Connectivity (JDBC)

The Java Database Connectivity(JDBC) is a set of APIs that lays the framework for allowing Java programs to connect to a database. It is the mechanism by which JAVA interacts with a database. It is a low-level interface for making queries and transactions to the database. JDBC does this task by performing three important functions. First, it establishes a connection to the database via one of the drivers, JDBC addresses a database with a URL-like syntax which is convenient for databases accessed over the internet; Second, it sends SQL statements to the database by way of a driver, contrary to popular belief, JDBC does not translate standard SQL statements into vendor specific dialects. Furthermore, JDBC calls are not checked for semantic, syntactic, or schema correctness at compilation, these errors are

only seen at runtime; Third, JDBC processes results from the database back into Java format, these capabilities allows JAVA to transact with various databases.

3.5.1 JDBC for Distributed Environments:

JDBC is suitable for modern computing environments, most these environments are distributed. This means that each of the participants in an environment contributes a small amount computing. Client/server (C/S) architecture is an example of a distributed environment. At its core level, C/S is very simple to understand. The client is the resource taker and the server is the resource giver. This does not mean that the client cannot create new information to be sent to the server. In most cases, many clients connect to one server by JDBC. The server is typically the resource repository for all the data (hence the term database). Furthermore, when C/S is used over a network, a client can be in another room, state, or country from its server. In other words, the client can connect the server from any remote locations by JDBC across the networking.

3.5.2: JDBC Driver Classification:

There are four kinds of JDBC drivers, the following is the four kinds of JDBC in detail.

1: Type 1--JDBC/ODBC Bridge:

ODBC (open database connectivity) is Microsoft's attempt to create a mechanism that will protect programmers from the differences in various vendor specific SQL dialects [CHAS00, UNGLA00]. Java took advantage of Microsoft's earlier work by providing a path for Java to interact with ODBC. ODBC is the Windows API standard. It is based on the X/Open Call-Level

Interface (CLI) specification, a standard API for database access. This move of creating a JDBC/ODBC bridge immediately provided access to all the databases that provide an ODBC driver. This manner of interacting with the database has received some criticism, however, performance is allegedly decreased. Furthermore, ODBC is not programmed in Java, but in C, therefore compromising Java's platform independence and security features. Lastly, some native code must be loaded on the client side for this option [LIND00, JDBC01].

2: Type 2--Java to native API (Part Java, Part Native Driver)

The second kind of driver is the Java to native API. This type of driver translates JDBC calls into calls for the specific database via a vendor's API. This option is generally not a pure Java solution, however. This limitation implies that platform specific code must be developed on the client side. It is, however, the simplest option for database vendors who want to create a driver quickly [CHAS00, LIND00, JDBC01, MSQL01].

3: Type 3—Net-Protocol Java Driver(Intermediate Database Access Driver)

Thirdly, there is the net-protocol Java drivers. The net-protocol option is a middleware solution where the JDBC calls are translated into a DBMS independent net-protocol. The net-protocol is then handled by a server which makes the database specific calls via middleware products communicating to the DBMS. The main advantage of this solution is that databases are accessible to any machine that has the Java Virtual Machine (JVM) installed [CHAS00, LIND00, JDBC01, MSQL01].

4: Type 4—Pure Java Driver:

Native-protocol pure Java drivers are the fourth kind of driver. The JDBC calls are translated into a network protocol, which is then directly interpreted by the DBMS, instead of being handled by a middleware product. Similarly, the native-protocol drivers run on any machine that have a JVM [CHAS00,LIND00, JDBC01,MSQL01].

3.5.3: The System Cost of JDBC:

The following is the system cost for JDBC database access:

A. Initialization phase:

1. The time for the application to establish connection to the database.

B. Execution phase:

1. The time for the application to issue an SQL statement to the database and obtain the results.

The limitation of JDBC is: It just supports the Java programming.

After discussion about system cost among different system architectures above, in order to understand which one is system cost effective easily, the following table gives a detailed analysis (The system cost includes two phases: Initialization phase and execution phase, initialization phase is the procedure for establishing database connectivity, and execution phase is the procedure for querying the database after database connection has established).

System Architectures	Cost of Initialization phase	Cost of Execution phase
CORBA	1. The time for the client to initialize core ORB classes. 2. The time for the client to bind to the application server.	3. The time for the client to invoke a method on the application server passing the SQL statement as a parameter. 4. The time for the application server to execute the SQL, obtain and return the results to the client.
RMI	1. The time for the client calls a bind method of RMI to obtain a reference to the remote application server (bind to it).	2. The time for the client to invoke a method on the application server passing the SQL statement as a parameter. 3. The time for the application server to execute the SQL statement, and then return the results.
Socket	1. The time for the client to open a socket connection with the application server.	2. The time for the client to pass to the application server the data stream containing the SQL statement. 3. The time for the application server to execute the query, obtain the results, and return them to the client
Servlets	1. The time for the client to open a URL connection with the Web server.	2. The time for the applet to invoke, through the Web server, the corresponding servlet passing the SQL statement as a parameter

		(stating explicitly the servlet name and type of operation). 3. The time for the servlet to execute the request, obtain and return the entire result table to the client.
JDBC	1. The time for the application to establish connection to the database.	2. The time for the application to issue an SQL statement to the database and obtain the results.

Table 1: System Cost Comparison

From the table above, we see that CORBA, RMI and Servlets need the middle tier support, middle tier accepts the calls from client and then transfers them into the database server calls. Obviously, the middle tier process is time consuming. The socket and JDBC are relatively cost effective. The socket can establish direct connection between client and database server, it is cost effective, but socket is lower-level network communication programming, we must write our own code to handle the incoming and outcoming data streams, the system will be too complex if we establish our system that depends on the socket completely. JDBC is another good choice, particularly using type 4 pure Java JDBC. On the first hand, it converts the client calls to direct network calls using vendor-specific networking protocols by making direct socket connections with the database [SUBRA00]. This is the most efficient method of accessing databases, both in performance and development time. On the other hand, all subsequent queries for JDBC require only the execution phase once the JDBC database connection is established.

So, based on the comparison, we chose to use JDBC for our system tool since it provides the most effective access. We did not do experiments for each

system architecture described above, and we did not get the measure of access cost for these systems; this work may be done in the future as part of continuing research focused on performance and design evaluation.

In the following, we give features of various JDBC types and their comparison in detail.

3.5.4: Features of 4 Types JDBC and Their Comparison:

- **Type 1:**

The JDBC-ODBC's bridge translates all JDBC API calls into equivalent ODBC calls, the driver then delegates these calls to the data source, the Java classes for the JDBC API and the JDBC-ODBC's bridge are invoked within the client application process [SUBRA00]. Similarly, the ODBC layer executes in another process, this configuration requires the client application to have the JDBC-ODBC's bridge API, the ODBC driver, and the native language level API (such as the OCI library for Oracle) installed on each client machine.

Due to the multiple layers of indirection for each data access call, this solution for data access is inefficient for high-performance database access requirements. Not only does the system have to pass the database call through multiple layers, but it also limits the functionality of the JDBC API to that of the ODBC driver. Using a bridge to an ODBC data source is not a preferred solution, but in some cases it might be the only solution. For instance, a Microsoft Access database can only be accessed using the JDBC-ODBC's bridge.

- **Type 2:**

The second alternative is the Type 2 driver. Type 2 drivers use a mixture of

Java implementation and vendor-specific native APIs for data access. This is similar to the Type 1 driver architecture, except that there is one less layer to go through and so it is much faster. When a database call is made using JDBC, the driver translates the request into vendor-specific API calls. The database will process the request and send the results back through the API, which will forward them back to the JDBC driver. The JDBC driver will format the results to conform to the JDBC standard and return them to the program.

In this approach too, the native JDBC driver (part Java, part native code) must be installed on each client along with the vendor-specific native language API. The native code uses vendor-specific protocols for communicating with the database. The improved efficiency makes this a preferred method over the use of Type 1 drivers and it also means that potentially we have use of the full functionality of the vendor's API.

- **Type 3:**

Type 3 drivers are based on intermediate (middleware) database servers with the ability to connect multiple Java clients to multiple database servers. In this approach, clients connect to various database servers via an intermediate server that acts as a gateway for multiple database servers. While the specific protocol used between clients and the intermediate server depends on the middleware server vendor, the intermediate server can use different native protocols to connect to different databases. The Java client application sends a JDBC call through a JDBC driver to the intermediate data access server. The middle-tier then handles the request using another driver (for example, a type 2 driver) to complete the request.

Architecturally, this is a very flexible alternative, as the intermediate server can abstract details of connections to database servers. J2EE application servers such as Weblogic include Type3 drivers.

- **Type 4:**

This is a pure Java alternative to Type 2 drivers. These drivers convert the JDBC API calls to direct network calls using vendor-specific networking protocols by making direct socket connections with the database (such as Tabular Data Stream for Sybase and Oracle Thin JDBC Driver) [SUBRA00]. This is the most efficient method of accessing databases, both in performance and development time. It is also the simplest to deploy since there are no additional libraries or middleware to install as shown overleaf. All major database vendors (Oracle, IBM, Sybase, and Microsoft, etc.) provide Type 4 JDBC drivers for their databases and they are also available from third party vendors.

After discussing and comparing the 4 types of JDBC, we can say that the type 4 pure Java driver's performance is the best. So, our system will use the type 4 pure Java driver, this pure Java driver will be embedded into our system's code provider component and code consumer component. They are both Graphic User Interfaces (GUI) of Java Swing, they can use the embedded pure Java driver to connect the remote centralized database directly. This is the most efficient method of accessing object-relational databases, both in low system cost-access performance and saving development time

3.6: Our Solution:

Based on the theory of chapter 2 and chapter 3, in this section, we give our solution for code services system.

In Chapter 2, we have discussed the code storage and retrieval, creating the user-defined data type including LOBs (CLOB and BLOB) within object-relational database to store the source code and corresponding binary

code is a very good approach. Since CLOB and BLOB are treated as objects, it is easy to retrieve and manipulate them, and also the retrieval is more effective.

In this chapter, we discussed several architectures for code service system and their performance. We found that the system cost of JDBC is the most effective among several architectures, and the Java Jini is very suitable for our code service retrieval system.

Our solution is to integrate the object-relational database technology, Java Jini concept and JDBC technologies together to establish: a centralized object-relational database based code service retrieval system tool for software reuse. In our system, we create user-defined data type including CLOB, BLOB to store source codes and corresponding binary codes. Centralized object-relational database is accessed by remote users, the centralized database is flexible, can be put in anywhere. We also use new pure Java client-side JDBC to access new data types including CLOB and BLOB in our system. This system is cost-effective and access-effective.

In Chapter 4, we will describe our system's design and implementation in detail.

Chapter 4: System Prototype Design and Implementation

4.1 System Overview:

At the end of the chapter 3, we gave our solution for code services retrieval system, that is to integrate the object-relational database technology, Java JINI concept and JDBC technologies together to establish: A Centralized Object-Relational Database Based Code Services Retrieval System for Software Reuse. Based on the theory of both chapter 2 and chapter 3, in this section, we discuss our system prototype design and implementation in detail.

In my thesis project system, we would like to solve two problems: one is the code's storage, the other is establishing a cost-effective, easy-to-use code services system architecture.

About code storage, our system creates the user-defined data types including LOBs (CLOB and BLOB) within object-relational database to store the source codes and corresponding binary codes. It is a very good approach, on the other hand, because CLOB and BLOB are treated as objects in our system, so, we can use object-oriented programming based approaches to retrieve and manipulate these objects; it improves the code retrieval effectively.

About system architecture, our system just use the concept of Java Jini, Jini uses component called lookup service to store the services. Usually the users of code service system are located on remote computers, they do not know where the codes are. Moreover, the amount of source codes and corresponding binary codes are large. However, Java Jini just only support to store small amount of services. So, we need a centralized place to store the large amount of codes and users themselves information, based on this requirement, we think that establishing a centralized database system architecture is good

approach in our system. The object-relational database is the centralized place, which is flexible. It can be put on a powerful server which locates on anywhere in the world. The users are distributed, they can access, and retrieve information from the centralized database from any remote location by two components: Code Provider Component and Code Consumer Component that are offered by our system.

The two components use embedded new pure Java client-side JDBC to access the centralized object-relational database because this connection is the most efficient method of accessing object-relational databases in low cost-access performance.

Both code provider component and code consumer component are easy-to-use and have friendly Graphic User Interfaces (GUIs) which support code's collaborative programming and code reuse, they both are programmed by Java and can be run on almost every platform which only install Java Runtime Environment (JRE).

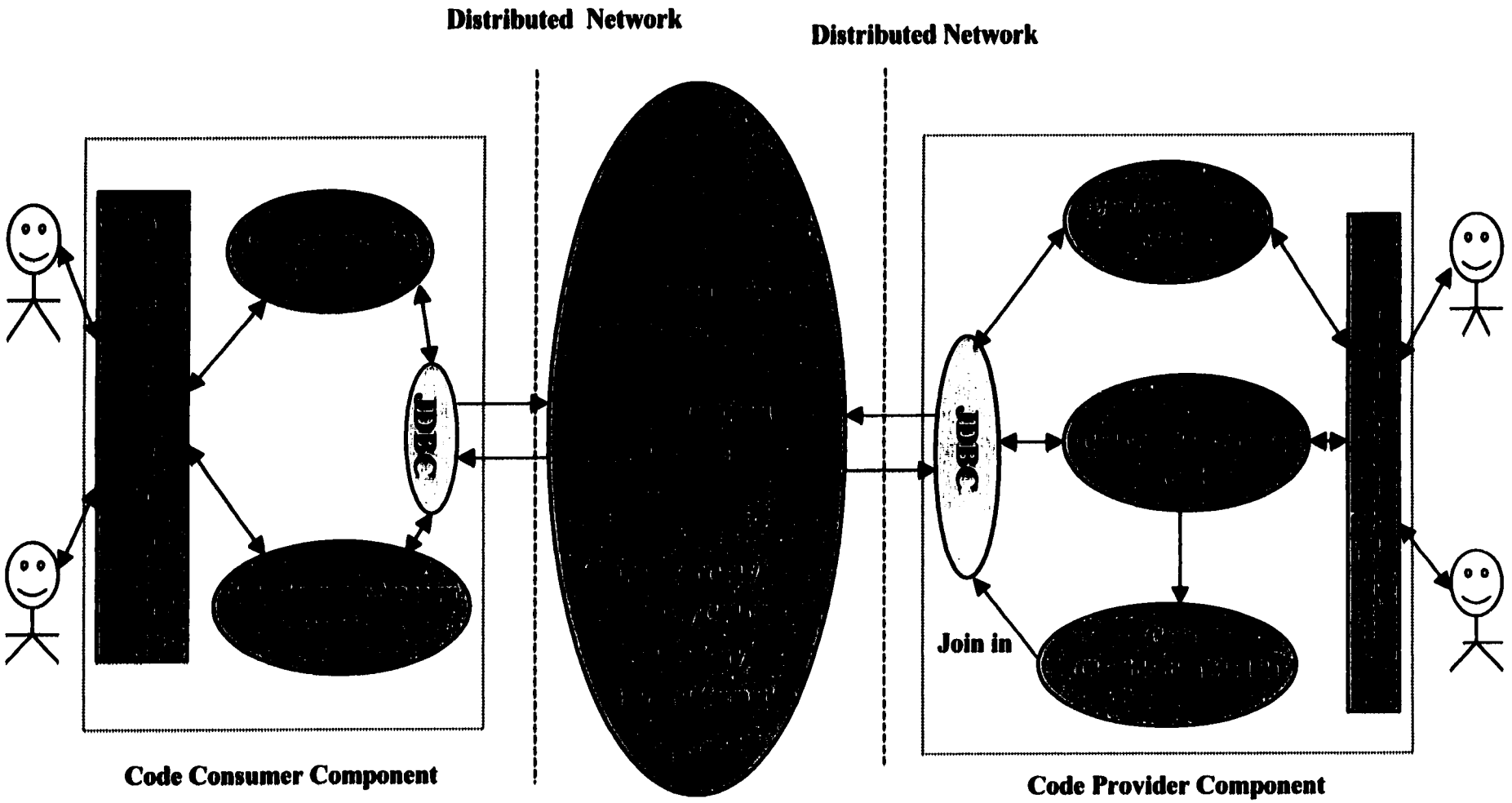
Code provider component is used by the code providing users. They can insert, build, update, maintain the computer language source codes and corresponding binary codes from remote locations; Code consumers component is used by code consuming users, they can query, access, retrieve the computer language source codes and corresponding binary codes from remote locations. All the transaction operations of both components are processed on a distributed networking environment.

Code provider component also implements the basic functionalities how to query/search and classify the code services because they are necessary and essential components for code service system.

4.2: System Architecture:

In this section, we discuss our system architecture in detail. Our code service retrieval system has four main components: Centralized Database Component; Code Provider Component; Code Consumer Component and Pure Java JDBC Component. Centralized database component should be deployed on a powerful computer which can handle a large amount of user's accesses; Code Consumer Component and Code Provider Component can be downloaded by users who want to use our system, both components can be run on user's PC or laptop that have installed Java Runtime Environment (JRE); Pure Java JDBC Component is embedded inside both code consumer component and code provider component. **Figure 1** shows the whole architecture of our system.

System Architecture



- **Figure 1: System Architecture**

1. Centralized Database Component:

Centralized Database Component is the key part of our system. It is on the middle of the **Figure 1**, it is implemented by object-relational database which offers a centralized place to store the large amount of codes and users themselves information. It is a code repository, it is a good place to store source codes and corresponding binary codes by creating our own User-defined Data Types. The new User-defined data types including Large Object data types such as Character Large Object (CLOB) data type and Binary Large Object (BLOB) data type are used to store our source codse and corresponding binary codes. We use Character Large Object (CLOB) data type to store source codes such as Java, C++ and C, and use Binary Large Object (BLOB) data type to store executable corresponding binary codes and other related binary files. On the other hand, it also stores the code provider and code consumer users' information. The users who locate on different remote computers can use code provider component and code consumer component to access, query and retrieve the centralized database from any remote location by Java pure JDBC. It should be ported on a powerful computer that has high performance computing, and can handle huge access requirements. It also supports the code integrity, transaction, concurrency control and code data recovery.

2. Code Provider Component:

The code providing users use this component to access the remote centralized database component, they insert, build, update, maintain the computer language source code and corresponding binary code and other related files from remote locations. It consists of three modules: **User Register Module, Code Provider Manager Module and Code Classification Module.**

- **User Registration Module:**

This module offers two system functional parts. One is the code provider user register functionality, and the other is code provider user authentication functionality. Both are code services level access, our code service system done not use the system level access. In other words, both the user register part and user authentication are supported by our system level, not by system Level.

The first one is used to allow user to register themselves, code provider users can input their name, preferred userid, password, email and address into the centralized database, this part also asks user to choose which membership he/she belongs to such as Sharcnet network member, Canada C3 network member or just general member. They can also change their register information on-line after input; Code provider user authentication part offers a basic system security functionality. In order to use the code provider component to access the centralized database, the code provider user must input his/her correct userid and password, otherwise, the system will refuse him/her to use the code provider component; This module will also assign different user privilege to the user according to his/her membership after user register automatically, different memberships such as Sharcnet network member, Canada C3 network member or just general member has different user privileges, the user privilege is used to control the centralized database's access right.

- **Code Provider Manager Module:**

This module is very important component in our system, code provider user can do most of operations here, it offers seven system functional parts which are: code search/query part, view code detail part, insert code part, modify code part, delete code part, list rest code part and download code part.

1) Query/Search Part:

Code provider user uses this part to query, search the centralized code database. Sometimes they want to know the existing codes inside the centralized code database before they want to insert their own new codes, or want to find their own codes that have been inserted before. They can use this part to find the codes by offering keywords such as code name and code type, the result will be returned in a relatively short period time. When the amount of codes in the centralized code database are large, the list of codes will be very long, so, it is not convenient to find the code you want in the long list, therefore, it is a good idea to use the search/query part to locate the code you want directly.

2) View code detail part:

Code provider users can use this part to view the code features, this part displays the code attributes in detail such as code name, code language type, code compiler type, code executable platform, code author, code version and code description. The users can use these information to know the code they select in detail, and then make further decision such as making a decision if this code is useful for them, making a decision if this code is needed to modify, or making a decision if this code is needed to downloaded.

3) Insert code part:

Code provider users can use this part to insert their own source codes and corresponding binary codes into the remote centralized database, this part will call the code classification module automatically and the users must specify their own code's features (code attributes) such as code name, code language type, code compiler type, code executable platform, code author, code version and code description within code classification module before inserting them, then it will join in (save) all source codes, corresponding binary codes and

specified code's feature(code attributes) into the remote centralized database. This part also provides an editor that allow user to edit, modify their source codes.

4) Modify code part:

Code provider users can use this part to modify their source codes and corresponding binary codes which have been inserted into the remote centralized database before, it is an on-line modification if you want to change the code's features (code attributes). Similarly to insert code part, this part will call the code classification module automatically and let you modify the code's features, then it will join in (save) all modified source codes, corresponding binary codes and code's feature (code attributes) into the remote centralized database. This part also provides an editor which allow user to edit, modify their source codes.

5) Delete code part:

This part can be used by code provider users who have the deletion privilege to delete the source codes and corresponding binary codes from remote centralized database.

6) List rest code part:

A complete executable program code of Java, C++ or C should have three kinds of files: main code file, dependent code files and binary code files. Usually the code provider manager module just displays the main code. This part is used to display the rest of code parts such as dependent code and binary code. Please note that main code could have no dependent code sometimes. In this case, it just displays the binary code. Here, the "display" means that this part just displays the names of main code, dependent code and binary code, not the actual code. The users must download the main code, dependent code and binary code if they want to get the actual codes.

7) Download code part.

After viewing the code detail by using the view code detail part, the code provider user may find some codes that are useful for them and want to get the actual codes, they can use this part to download the actual main codes, dependent codes and binary codes from remote centralized database.

- **Code Classification Module:**

Generally, codes are large and unstructured data, therefore, this module is called by code provider manager module to classify the codes. Before the codes are stored into the centralized database, they are classified based on classification and retrieval methods such as descriptive classification and type-based retrieval. According to the description of the code, we use this module to specify some attributes to the code such as code name, code feature, language type, compiler type, executable platform and name of author; We can also use type-based retrieval method to classify the different types of codes into Java, C++ or C. After code classification by this module, the unstructured codes are converted into structured data and then are joined into the remote centralized database, so that they are easy to be queried by keywords quickly.

3. Code consumer Component:

The code consuming users can use this component to access the remote centralized database component, they can browse, search/query the remote centralized database and download source code and corresponding binary code they want to use. It consists of two modules: **Client Register Module, Code Consumer Manager Module.**

- **Client Registration Module:**

This module is similar to the user registration module of code provider component, it also provides two system functional parts, one is the code consumer user register functionality, the other is code consumer user authentication functionality. The first one is used to allow users to register themselves, they can also change their register information on-line after input; Code consumer user authentication part offers a basic system security functionality, if the code consumer user wants to access the remote centralized database, he/she must input correct userid and password, otherwise, the system will refuse him/her to use the code consumer component. This module will also assign different user privilege to the user according to his/her membership after user register automatically.

- **Code Consumer Manager Module:**

This module offers five system functional parts which are code search/query engine part, view code detail part, my editor part, list rest code part and download code part.

1) Query/Search Engine Part:

This is the key part of code consumer manager module, code consumer user use this part to query, search the existing codes inside remote centralized code database. They can input keywords such as code name and code type to search the codes.

2) View code detail part:

Code consumer users use this part to view the code features. This part allows the users to see the code attributes in detail. The user can know if the selected code is useful and valuable for him/her after seeing the code information in detail by using this part and then make a decision if this code is needed to

download.

3) List rest code part:

Code consumer user uses this part to get further information about dependent code and binary code. This part is used to display the rest of code parts: Dependent code and binary code.

4) Download code part.

After the code consumer users make a decision which code they want to download, they can use this part to download the actual main codes, dependent codes and binary codes from remote centralized database.

5) My Editor part:

This part offers an editor that is used by code consumer users to edit, view, modify any source codes they have download into their computer. The code consumer user may want to edit the downloaded code and add do some modification, the editor is offered for convenient.

4. Pure Java JDBC Component:

This component is embedded inside both code provider component and code consumer component. JDBC calls of component are translated into a network protocol, which is then directly interpreted by the remote centralized database, This is a pure Java JDBC. Thus driver converts the JDBC API calls to direct network calls using vendor-specific networking protocols by making direct socket connections with the remote centralized database. This is the most efficient method of accessing centralized databases, it is independent, we do not need to deploy any additional libraries in the client's side computer and install any application server as middleware.

4.3: System Cost:

Our system use both code provider component and code consumer component to access the remote centralized database. The two components locate on client's computer, and, the centralized database locates on remote database server. So, the main cost of system access is the communication between either code provider component or code consumer component and remote centralized database. The cost of communication between code provider component and remote centralized database is same as the cost of communication between code consumer component and remote centralized database.

The following is our system's cost that equals the communication cost between code provider component and remote centralized database. It is same as the cost of communication between code provider component and remote centralized database.

A. Initialization phase:

1. The time for the code provider component to establish connection to the remote centralized database.

B. Execution phase:

2. The time for the code provider component to execute a JDBC SQL statement to the remote centralized database and obtain the results.

It just needs two steps, so the cost is most effective comparing with other architectures discussed in chapter 3. All subsequent JDBC SQL queries require only the execution phase once the JDBC database connection is established.

4.4 System Development Environment:

Based on the code storage and system architecture, we choose the following system development tools and running platform.

1: System coding Language:	Java programming language(Java 2, Sun JDK1.3.0).
2: Object-relational database:	Oracle corporation's Oracle8i.
3: Development platforms:	Microsoft Windows98/2000 and Linux, oracle8i for Windows98/2000, and for Linux.
4: Running platforms:	MS Windows95/98/ME/NT/2000, Linux, UNIX, Oracle8i database server, Java Runtime Environment(JRE).

- **Java Programming Language:**

Code provider component and code consumer component are two main components in our system. The two components will be downloaded by remote users into their computers, the users will use these two components to communicate with the remote centralized database. Since different users install different operating system on their computer such as MS windows, Linux, UNIX, in order to guarantee that the two components can run on these different platforms, therefore, the two components should be platform independent. So, Java can satisfy our requirement, Java is platform independent and its compiled class can be executed on almost any existing computers virtually; it is also a good programming language for distributed system based on networking; and, it supports object-oriented programming (OOP). Java was designed to be a platform independent language. The Java source code is translated into byte code by the Java compiler. The byte code at

this point can be interpreted on any machine provided that it has a JVM installed on it [CHAS00]. The Java VM sits between the Java program and the operating system, offering the program an "abstract computer" that executes the Java code and guarantees certain behaviors regardless of the underlying hardware or software platform. Java compilers do not produce assembly language for a particular machine but rather a platform neutral "byte code" that the machine-specific VM interprets on the fly[ENOS99]. From the database perspective, Java's portability features are attractive. The importance of platform independence will usually override discrepancies in the appearance of a Java program over various platforms. Database software developers will certainly not overlook these portability issues [NIEM00]. Java Database Connectivity (JDBC) is the mechanism by which JAVA interacts with a
database, it is a low-level interface for making queries, and transactions to the database.

Our system is based on the object-oriented programming, every thing in our system is treated as object, therefore, we have used a lot of object-oriented programming's features in our system. Actually, we create user-defined data type to store our codes, this data type is object, object-relational database also add the object-oriented programming's features into the relational database. Java is good language for object-oriented programming, with respect to its object-oriented capabilities, Java was designed to be object oriented from the ground up, based on the realization that in order to function within increasingly complex, network based environments, programming systems must adopt object-oriented concepts. Java technology provides a clean and efficient object-based development platform.

Security is another important issue for our system, the code provider component and code consumer component locating on user's personal

computer communicate with the centralized database across the distributed networking environment. Java can satisfy our system security requirement, since Java technology was designed to operate in distributed environments, security features were designed into the language and run-time system. Java technology lets you construct applications that cannot be invaded from the outside. In the network environment, applications written in Java are secure from intrusion by
unauthorized code such as viruses that may attempt to attack file systems.

- **Oracle 8i:**

In our system, we use object-relational database to implement the centralized database component. We create the user-defined data type including LOBs (CLOB and BLOB) within object-relational database to store the source code and corresponding binary code. Oracle 8i can meet our need.

Oracle 8i is the world' most popular database for distributed computing. Oracle 8i has brought the relational database world into the object relational area. The new features of the product make it possible to combine the newer distributed object-oriented structures with the traditional relational constructs. With Oracle 8i, computing services/resource can be stored into the Oracle. So, users can query/search the services very easily. Building and maintaining the system also should be faster, easier and with lower cost. Meanwhile, Oracle 8i add significant enhancement to keep pace with the technological requirements of demanding distributed and Internet applications. Through integrated Java Virtual Machine, Oracle 8i has improved performance to support for Java 2, and allow applications to efficiently implement and manage robust distributed application.

Oracle8i, the database for Internet computing, changes the way information is

managed and accessed to meet the demands of the Internet age, while providing significant new features for traditional online transaction processing (OLTP) and data warehouse applications. It provides advanced tools to manage all types of data, and it also delivers the performance, scalability, and availability necessary to support very large database (VLDB) and mission-critical applications. Oracle8i is much more than just a simple relational data store.

Oracle8i introduces new support for Java by including a robust, integrated, and scalable Java Virtual Machine (VM) within the server, named Oracle JServer. This expands Oracle's support for Java into all tiers of applications.

Oracle8i provides a number of object-oriented features that let us transfer our design and problem-solving skills from those languages to database application development.

The most widely accepted database model is the relational model. Oracle8i extends the relational model to an object-relational model, which makes it possible to store complex business models in a relational database. The object-relational model allows users to define object types, specifying both the structure of the data and the methods of operating on the data, and to use these data-types within the relational model. In our system, we can define additional kinds of data types, and use these data types to store our code services. Oracle8i supports object-relational features such as internal large objects and external large objects.

- **Platforms:**

Our system divides the platforms into two parts: Development platforms and running platforms. Development platforms are the platforms on which we

develop our system's components, we installed three operating systems on my computer which are Microsoft Windows98, Microsoft 2000 and Red Hat Linux7.0; We also installed oracle8i object-relational database on each of the three platforms. So, We have three separate systems on my computer. Usually we program on the Windows 98 first, and if the code runs well, then we run the same code on Windows2000 and Linux system and make sure the same code give no problem on the other two platforms. If we get problems, we must debug it and re-test until no problems for the three platforms, the reason why we program the code on three different platforms is because our system's users use different platforms, we can not predict which platform the users use in advance, so our system's components must be needed to run on most popular platform well.

Running platforms are the platforms on which our system's components will run finally, they should include most popular platforms such as Windows95/98/ME/NT/2000, Linux, UNIX.

4.5 System Design Specification:

In this section, we will specify our system design by using Unified Modeling Language (UML), it is a good system design modeling language, it is a language that unifies many of the industries' best engineering practices for modeling system. Based on the object-oriented paradigm. UML is used here for specifying, visualizing, constructing and documenting systems [BOOCH99]. It is the best choices for building the distributed object-oriented system since every computing service is treated as an object in our system. It is independent of particular programming language and development processes, and it can support higher level system design concepts such as collaborations, frameworks, patterns and component.

- **Use Case:**

Use case specifies the behaviors of our system, the code provider component, and code consumer component. It is a description of a set of sequences of actions. Next we take a look at the use case of our system, the code provider component, and the use case of code consumer component.

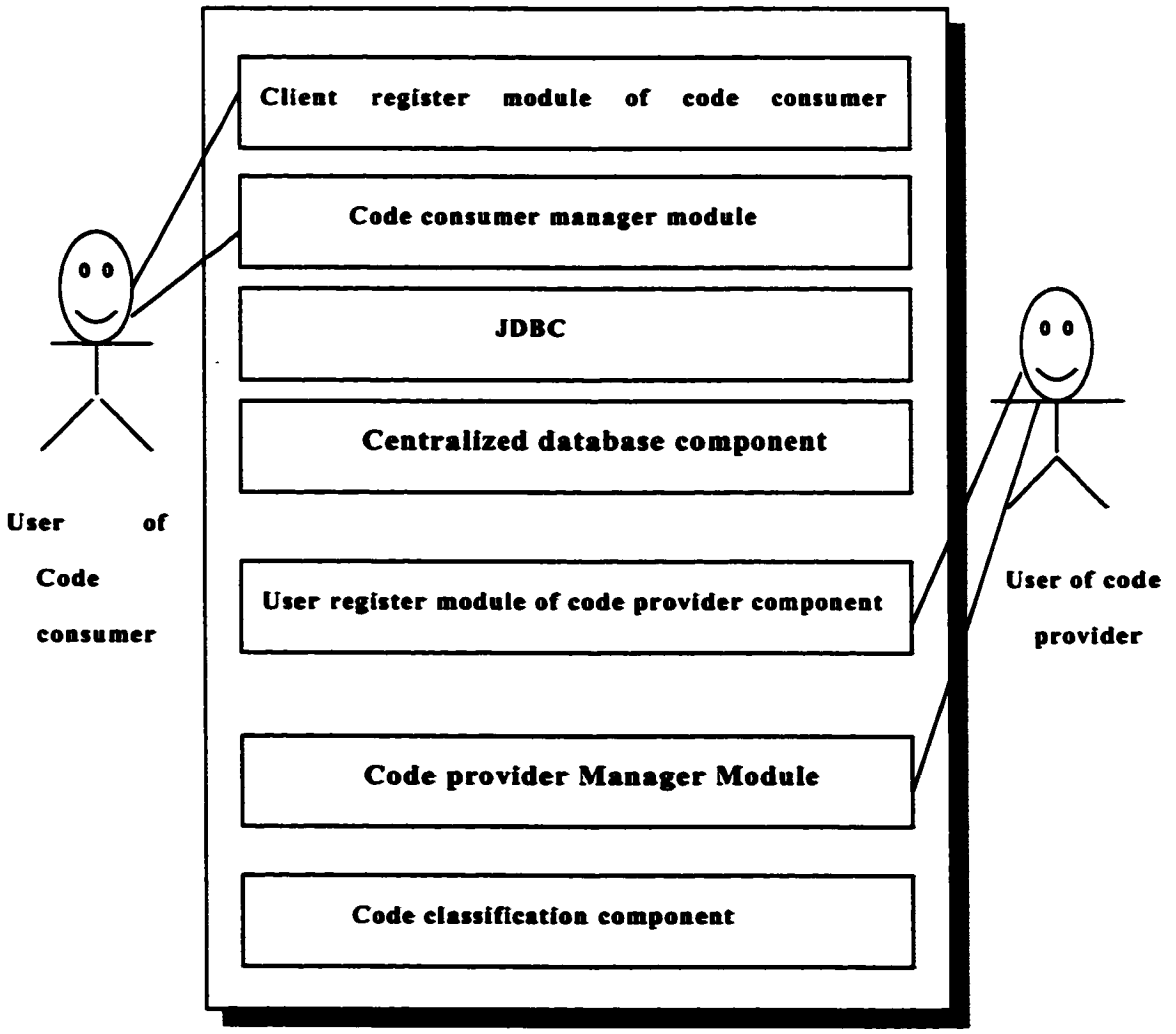


Figure 2: System Use Case Diagram

The above **figure 2**, it shows the system use case diagram, this use case describe the whole system's functionalities, the system has seven parts: client

registration module of code consumer component, code consumer manager module, Java JDBC, centralized database component, user registration module of code provider component, code provider manager module and code classification component; The users of code consumer component can use client registration module of code consumer component to register, and use code consumer manager module to search/query and download the codes from remote centralized database; The users of code provider component can use user registration module of code provider component to register, and use code provider manager module to search/query, insert, modify, delete and download the codes from remote centralized database. Code classification component is used to classified codes.

From above system use case, we know that the users cannot communicate with the centralized database component directly, they must communicate with the centralized database through JDBC component. Firstly, the client registration module of code consumer component and code consumer manager module communicate with the JDBC component, and then the JDBC component is used to access the centralized the centralized database; Similarly, the user registration module of code provider component and code provider manager module also communicate with the JDBC component, and, the code provider manager module also communicate with code classification module. Then finally, the JDBC component is used to access the centralized the centralized database. So, the JDBC is a bridge in our system.

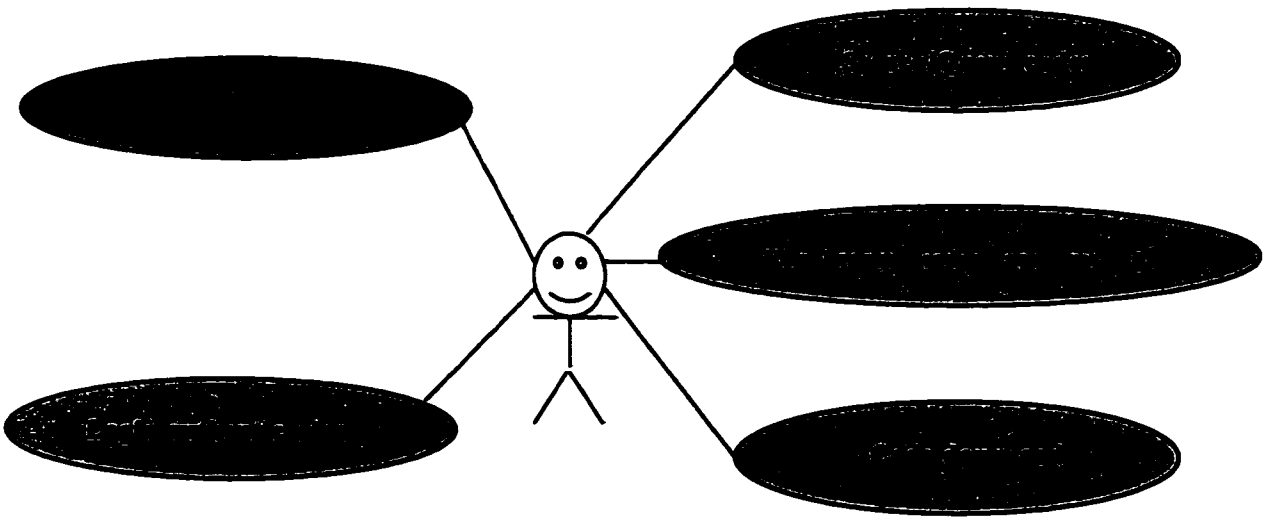


Figure 3: Code Consumer Component Use Case Diagram

Above **Figure 3** is the code consumer component use case diagram, this use case specifies the functionalities of code consumer component in detail, the users of code consumer should register themselves before using this system; The user must use login authentication part to input correct userid and password to login, otherwise, the system will refuse him/her to use the code consumer component; The users use search/query part to query, search the existing codes in the remote centralized code database, they can input keywords to search the code database; The user also uses view/browse part to view the code features they select. This part allows the users to see the code attributes in detail and then to make a decision whether they need to download it. This part also allows users to view the rest of codes such as corresponding dependent codes and binary codes. After the users make a decision which code they want to download, they can use codes download part to download the actual main codes, dependent codes and binary codes from remote centralized database.

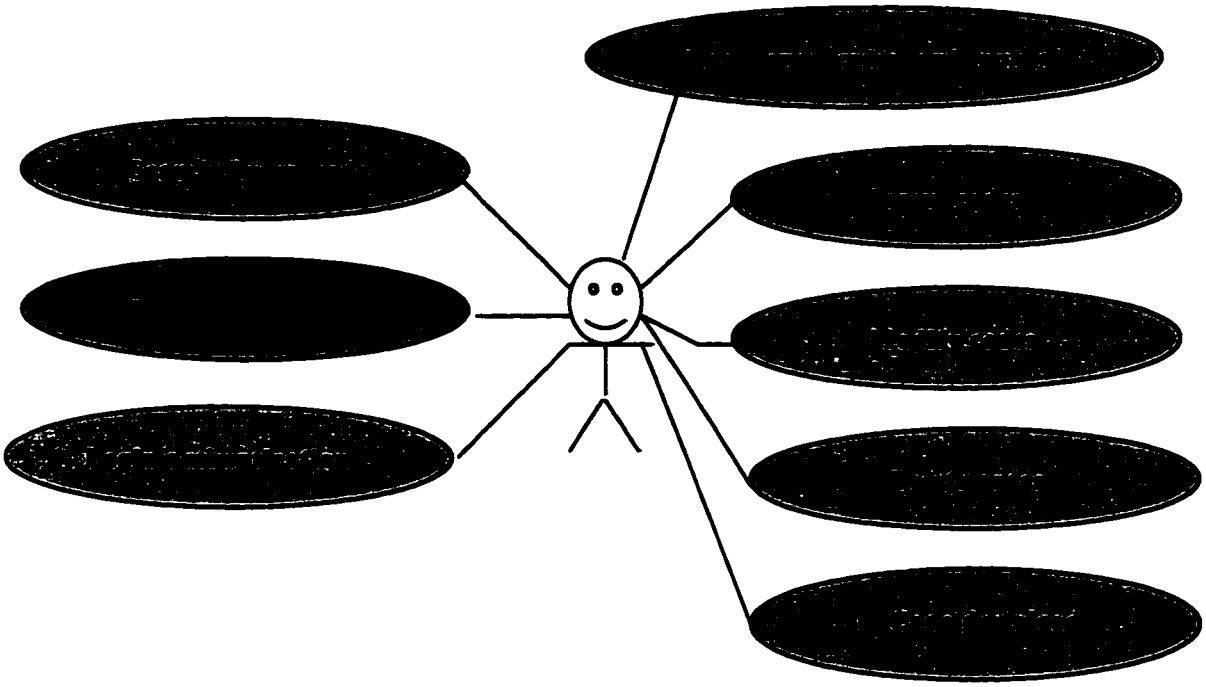


Figure 4: Code Provider Component Use Case Diagram

Figure 4 above illustrates the code provider component use case diagram. This use case specifies the functionalities of code provider component in detail. It is similar to code consumer component. The users of code provider should register themselves before using this system by using user registration part. The user must use login authentication part to login by inputting correct userid and password. The user uses search/query part to query, search the existing codes of remote centralized code database, the user can input keywords to search the code database. The users also use view/browse part to view the code features they select, this part also allows users to view the rest of codes such as corresponding dependent codes and binary codes. The users can use insert code part to insert their own source codes and corresponding binary codes into the remote centralized database, this part will call the code classification module that classifies the codes by using classification and retrieval methods such as descriptive classification and type-based retrieval

automatically. The users can use modify codes part to modify their source codes and corresponding binary codes which have been inserted into the remote centralized database before. The delete codes part can be used by users who have the deletion privilege to delete the source codes and corresponding binary codes from remote centralized database; After the users make a decision which code they want to download, they can use codes download part to download the actual main codes, dependent codes and binary codes from remote centralized database.

4.6 System Implementation Details

In this section, we will discuss the implementation of our system in detail.

- **Implementation of Centralized Database Component:**

In chapter 2, we have discussed the code storage. Creating new user-defined data types including large object such as CLOB and BLOB is good approach to store the source codes and binary codes. In our centralized database, we define our own five data types, three of them contain CLOB and BLOB to store the source codes and corresponding binary codes, and then define five tables by using these five user-defined data types. The following are the five our own defined data types (detailed implementation can be found in APPENDIX A).

1)**CREATE TYPE** CONSUMER_USER_TABLE_TYPE **AS OBJECT**
{Attributes .... };
2)**CREATE TYPE** PROVIDER_USER_TABLE_TYPE **AS OBJECT**
{Attributes .... };

These two data types define the code consumer users and code provider users' attributes such as name, userid, password, address, user privilege and so on.

3) **CREATE TYPE** MAIN_CODE_TABLE_TYPE **AS OBJECT**
{Attributes .... };
4) **CREATE TYPE** DEPENDENT_CODE_TABLE_TYPE **AS OBJECT**
{Attributes .... };
5) **CREATE TYPE** BINARY_CODE_TABLE_TYPE **AS OBJECT**
{Attributes .... };

The above three data types define the attributes of main code, corresponding dependent code and binary codes. They all contain the Large Object data types such as CLOB data type and BLOB data type.

Then we can create tables in the centralized database by using the new data types just defined above. The following is the five tables' definition (detailed implementation can be found in APPENDIX A).

1)**CREATE TABLE** CONSUMER_USER_TABLE **OF**
CONSUMER_USER_TABLE_TYPE{ Constraints...};
2)**CREATE TABLE** PROVIDER_USER_TABLE **OF**
PROVIDER_USER_TABLE_TYPE{ Constraints...};
3)**CREATE TABLE** MAIN_CODE_TABLE **OF**
MAIN_CODE_TABLE_TYPE{ Constraints...};
4)**CREATE TABLE** DEPENDENT_CODE_TABLE **OF**
DEPENDENT_CODE_TABLE_TYPE{ Constraints...};
5)**CREATE TABLE** BINARY_CODE_TABLE **OF**
BINARY_CODE_TABLE_TYPE{ Constraints...};

Our five tables above are totally different from the tables of traditional database. They are tables of object-relational database, and so, are all object tables. So, for each object table of five tables, each column of the table maps to the each column of the data type, which we defines above.

- **Using Pure Java JDBC Driver.**

Our system's code provider component and code consumer component use the pure Java JDBC driver directly. Our system two components' JDBC calls are translated into a network protocol, which is then directly interpreted by the centralized database, rather than being handled by a middleware server such as web server or application server, so it is cost-efficient. Therefore, our system's two component will be run on different platforms by users finally, this type of pure Java JDBC driver can be run on almost any machine that have a Java runtime environment (JRE), JRE is very easy to be downloaded into user's computer.

This driver converts the code consumer component and code provider component's JDBC API calls to direct network calls which use vendor-specific networking protocols by making direct socket connections with the centralized database. In our system, we use Oracle8i database as centralized database. So, this way is the most efficient method of accessing databases in performance and development time.

- **Using Prepared Statement Object to Improve Database Access**

Since our system's code consumer component and code provider component use many SQL to access the centralized database, and usually the same SQL statement is executed a lot of times with just different parameters, we need to find a method to reduce the cost of executing the same SQL many time. Using prepared statement object is a good solution. A prepared statement object can hold precompiled SQL statements and allows us to precompiled our SQL and run it repeatedly. This method is more efficient to access the database since processing SQL strings is a large part of a database's overhead, getting

compilation out of the way at the start can significantly improve the database access performance. For example, in our system, the SQL INSERT statement of insertCode( ) method is executed a lot of times when the user insert or modify the codes for remote centralized database. We can compile this statement before it is executed. Compilation of SQL statement is a time-consuming process that typically involves parsing of the statement, binding with the tables and columns, optimization, and code generation, so, it is better to use a prepared statement, with prepared statement, this compilation of SQL is done only once.

4.7: How to Use System and User Interface:

In this section, we will introduce how to establish our system and how to use the interfaces of code provider component and code consumer component.

4.7.1: Establishing the Centralized Database Component

Centralized database component is object-relational database that is used to store the code services. It is a code repository. During the system development, we establish Oracle8i object-relational database on my Windows98 as the centralized database, its TCP/IP networking configuration is: the IP address is: 127.0.0.1(domain name is: localhost); listening port is:1521; the networking services ID(SID) is ORCL. In order to test the system's portable, we also establish Oracle8i as the centralized database on Windows2000 platform and Red Hat Linux7.0 platform separately, the TCP/IP networking configurations for the two platforms are same as Windows98.

One of our system's feature is: the centralized database is flexible and portable, it can be port on any database server which supports object-relational database. After developing and testing the centralized

database component on three platforms in my computer, we move the centralized database into our computer science department's database server, this database server is running Oracle8i database on Sun Solaris Unix5.7, its TCP/IP networking configuration is: the IP address is:137.207.76.4(domain name is: goedel.newcs.uwindsor.ca); listening port is:1521; the networking services ID(SID) is cs01. The users can user our system's code provider component and code consumer component to insert, modify, query/search and browse the code services of this centralized database from remote location through TCP/IP networking. The users can locate anywhere in the world such as Toronto, New York or Beijing, thus, our system is truly distributed networking code service system.

4.7.2 Establishing a Web Page for Components Download:

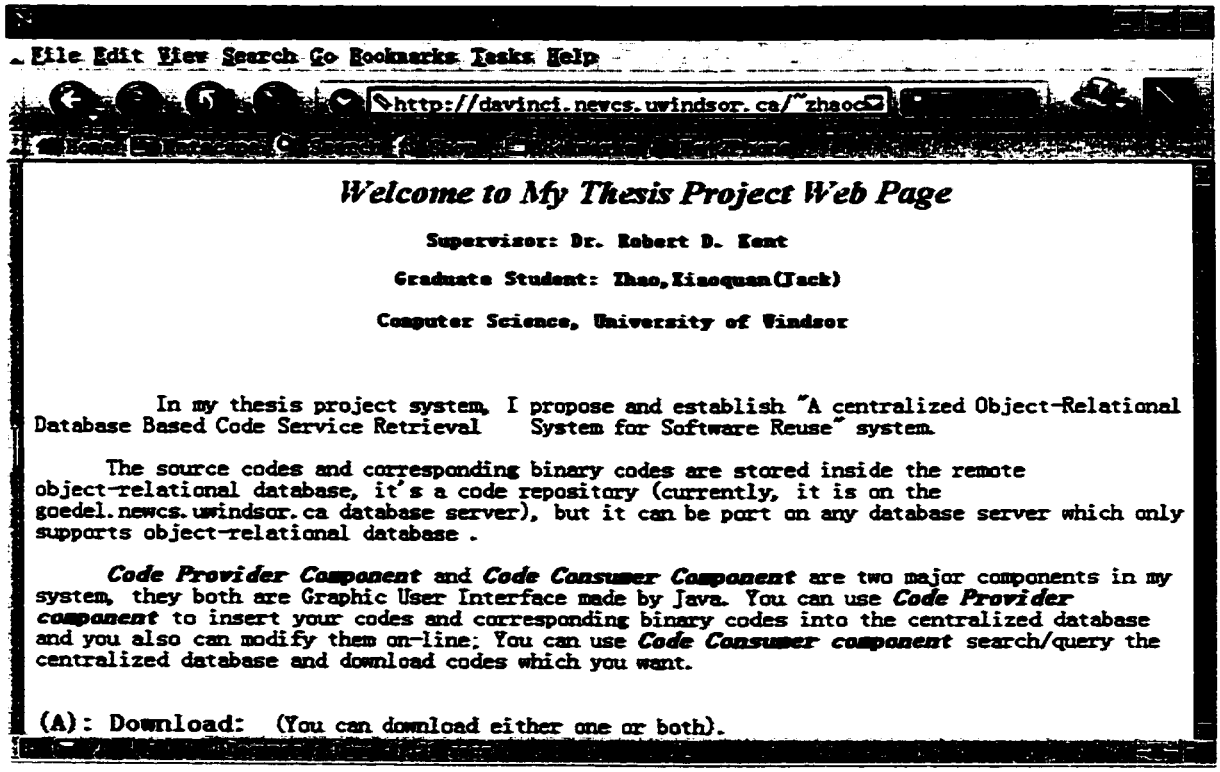


Figure 5: Part of Web Page for Components Download

Code Provider Component and Code Consumer Component are two major components in our system. The users can use Code Provider component to insert source codes and corresponding binary codes into the centralized database and they also can modify them on-line. The users can use Code Consumer component search/query the centralized database and download codes that they want. So, we establish a temporary's web page on our computer science department's web site **(Figure 5)** from where the user can download the two components from any remote location, the URL is http://davinci.newcs.uwindsor.ca/~zhaoc/. It is a temporary download web page for testing. The users can download either one or both by corresponding hyperlinks such as "Click me to download Code Provider Component" or "Click me to download Code Consumer Component", for IE user, just click the link to download directly, for Netscape user, right click and then click " Save Link As" to download.

- **System Requirements:**

After downloading the code provider component and code consumer component from this web page by users, they can run them on their computer, the following is the system requirements to run the two components well.

1) Windows95/98/ME/NT/2000/XP, LINUX, UNIX.
2) Java Runtime Environment(JRE).

Because the two component are compiled into Java .jar executable files, so the users do not need to make any configurations on their computer such as CLASSPATH set. Each of the two components includes all needed classes for runtime, all necessary classes are packaged into the java .jar files; The users just need to install the Java Runtime Environment(JRE) in their computers, if they do not have it, they can click hyperlink "Download JRE" to download the

JRE from this web page and then install it on their computers, they can run our two components after installing JRE.

4.7.3: How to Run the Two Components:

The following is how to run the code consumer component and code provider component after the users download them.

- **Various Windows Platforms:**

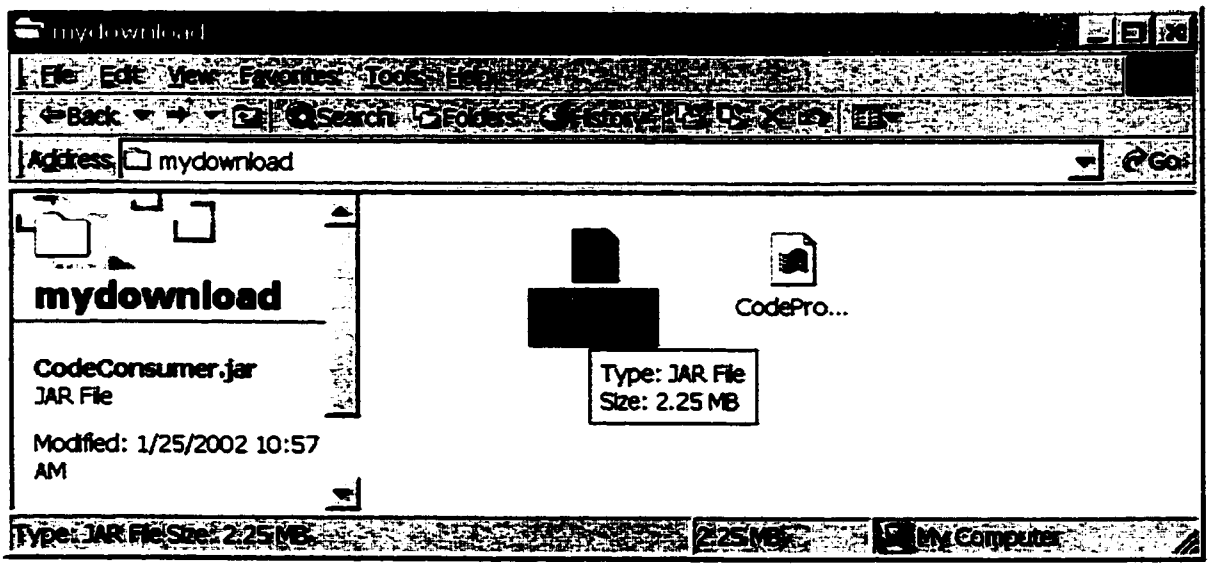


Figure 6: Run the Two Components On Windows Platforms

Figure 6 above shows how to run the two components on Windows Platforms, for example, after you download the two components from my web page and save them as "CodeProvider.jar" and "CodeConsumer.jar" executable Jar files. If you want to run the "CodeProvider.jar" file which is the code provider component to access the remote centralized code services database, just double click the "CodeProvider.jar", that is OK, this file will be executed on your computer automatically; Similarly "CodeConsumer.jar" file which is the code consumer component can be run.

Alternatively, you also can invoke DOS Shells by using "cmd" command, and then run the two components under Windows systems' DOS Shells, for this way, please open a DOS Shell and then change to corresponding subdirectory which saves the two components' files, and then use command "java -jar CodeProvider.jar " to run the code provider component or "java -jar CodeConsumer.jar " to run the code consumer component.

- **Various LINUX and UNIX platforms:**

Please open a C Shell or B Shell, and then change to corresponding subdirectory where you save the two components' files, and then use command "java -jar CodeProvider.jar " to run the code provider component or "java -jar CodeConsumer.jar " to run the code consumer component.

4.7.4: Code Provider Component's Interfaces

From above sections, we have discussed the how to establish centralized database component, how to download the code provider component and code consumer component, and how to run the two component on various operating systems. In this section and next section, we will show you how to use the two components' various interfaces in detail through diagrams.

1) Welcome Interface:

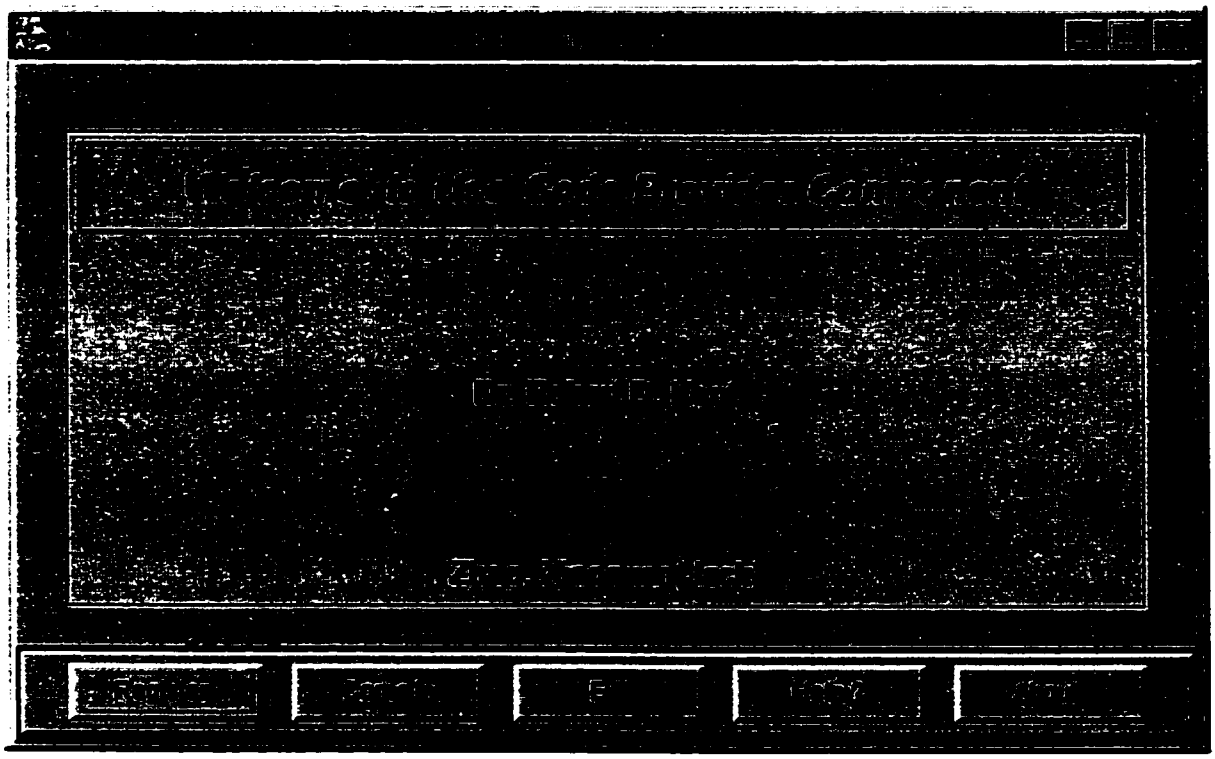


Figure 7: Code Provider Component Welcome Interface

Figure 7 above shows the first welcome interface after you run the code provider component. There are five buttons, <Register> button is used to register yourself before you want to user this code provider component. Press <Come in> button to enter the code provider if you already have registered and have valid userID and password. Press <Exit> button to exit this component, press <Help> button to get help about how to use this interface, press <About> button to get system information. If you click <Register> button, you will see the following **figure 8** code provider component user register interface.

2) Code Provider Component User Register Interface:

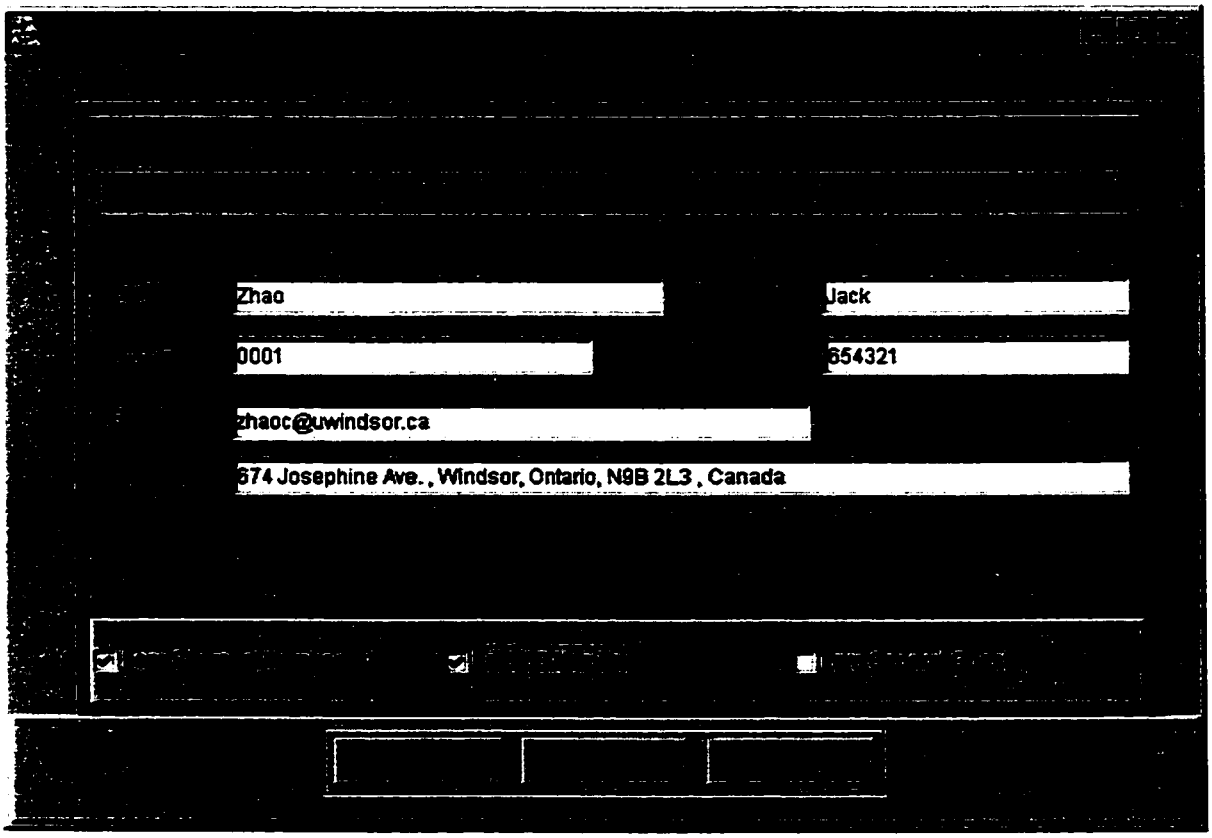


Figure 8: Code Provider Component User Register Interface

Figure 8 above shows the code provider component user register interface. This interface allows users to register themselves, the users can input their last name, first name, preferred userid and password, email, address. This interface also asks user to choose one membership which he/she belongs to from three options: Sharcnet network member, Canada C3 network member and just general member. This interface has three buttons: <Submit> button is used to store the user's information into the remote centralized database; <Reset> is used to re-fill in this register form, and, click <Cancel> to cancel this operation.

3) Code Provider Component User authentication Interface:

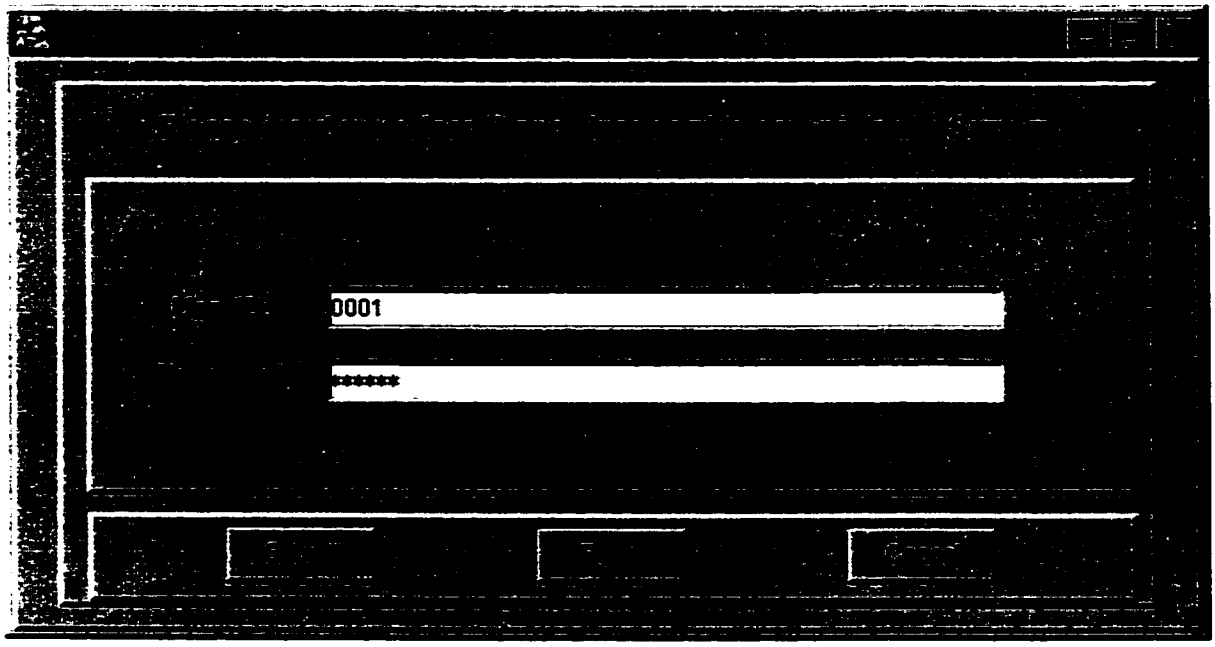


Figure 9: Code Provider Component User Authentication Interface

If you have already registered yourself, you can click the<Come in> button of **figure 7** to enter the system directly, the above **figure 9** will be popped up. You can input your userid and password and then click <Submit> button, your userid and password will be sent into the remoter centralized database for user authentication, if your information is authenticated successfully, the following figure 10 interface will popp up, otherwise a error message box will be popped up to give your some login error information. <Reset> button is used to re-fill in this authentication form; click <Cancel> to cancel this operation.

4) Code Provider Component User Operation Option Interface:

This **figure 10** interface allows user to do two operation options: either "Enter user account management component" or "Enter code provider component" by

clicking corresponding button. If you want to change your register information such as name, userid, password, address etc., you should click the button behind "Enter user account management component", the **figure 11** interface will be popped up. If you want to insert, modify, search/query the remote centralized code service database, you should click the button behind "Enter code provider component", the **figure 12** interface will be popped up.

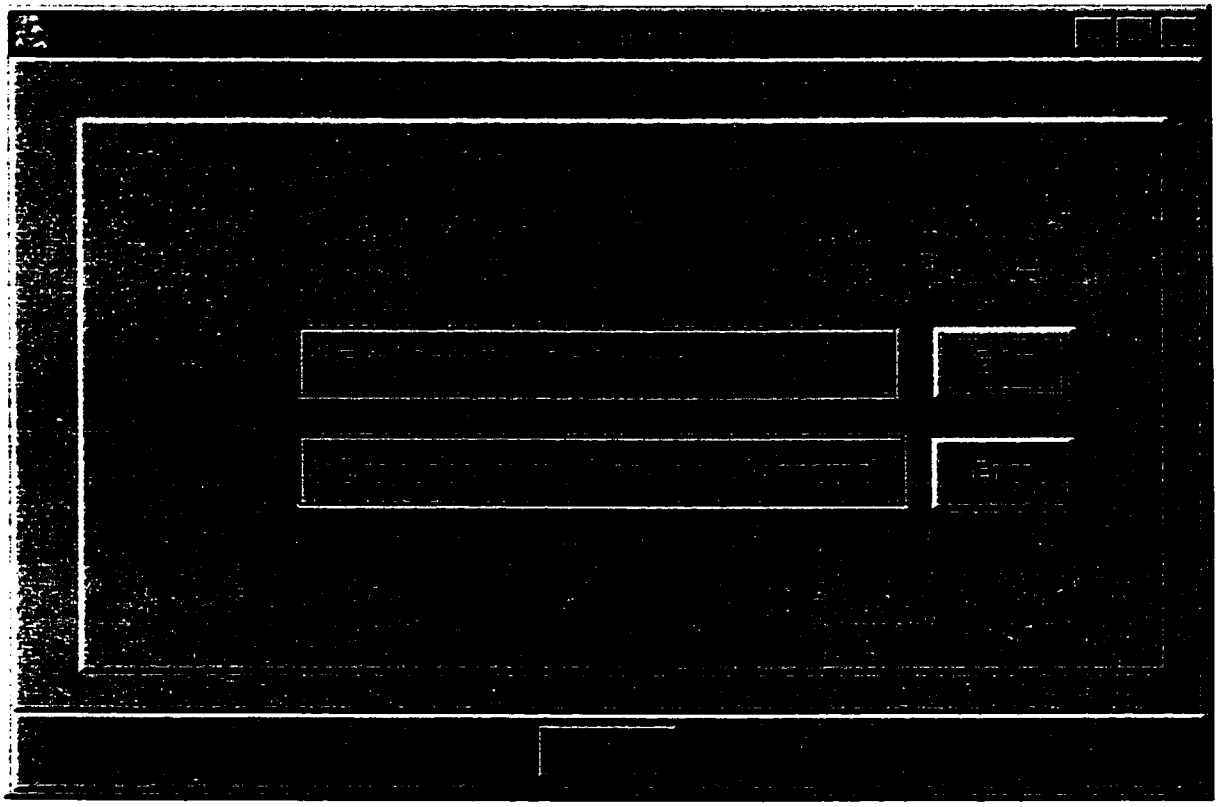


Figure 10: Code Provider Component User Operation Option Interface

5) Code Provider Component User Register Modification Interface

Figure 11 is the code provider user register modification interface. It is very similar to the interface of figure 8 above. This interface allows users to modify their register information on line, the users can modify their last name, first name, preferred userid and password, email, address, and membership

such as Sharcnet network member, Canada C3 network member and just general member.

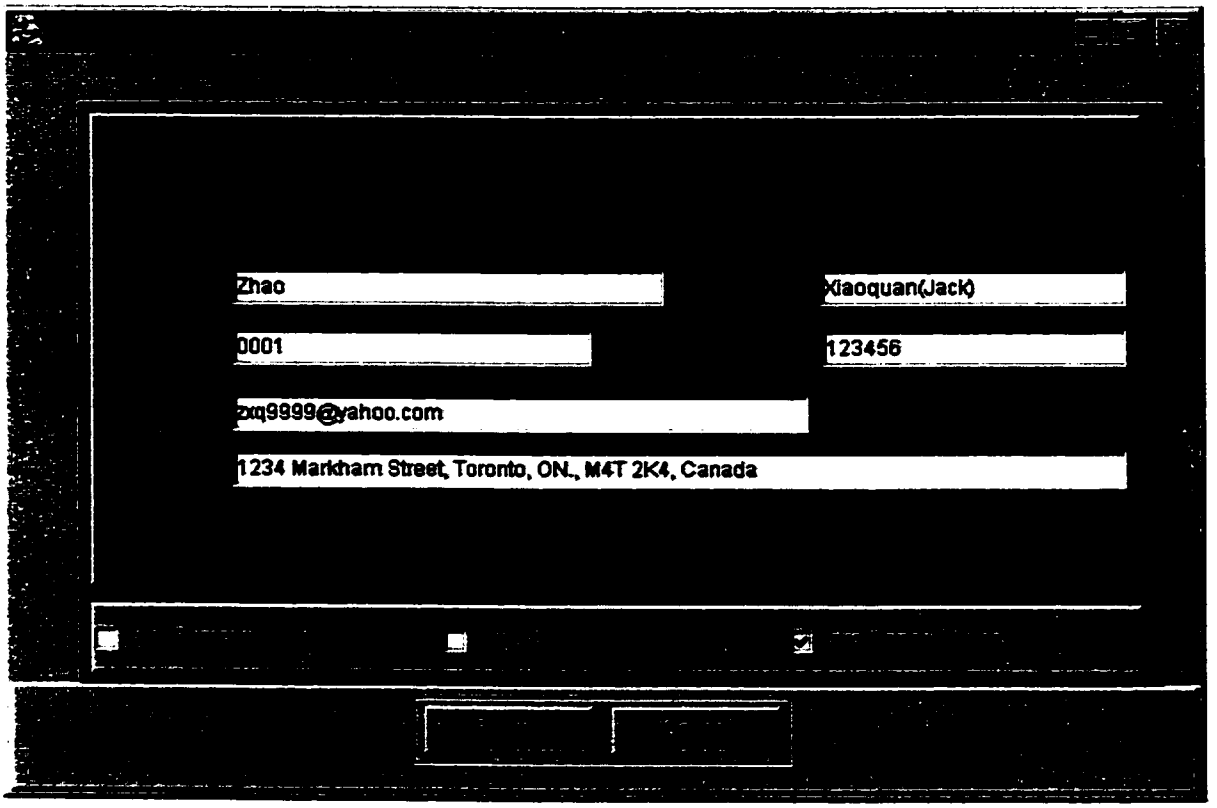


Figure11: Code Provider Component User Register Modification Interface

6) Code Provider Component User Main Operation Interface:

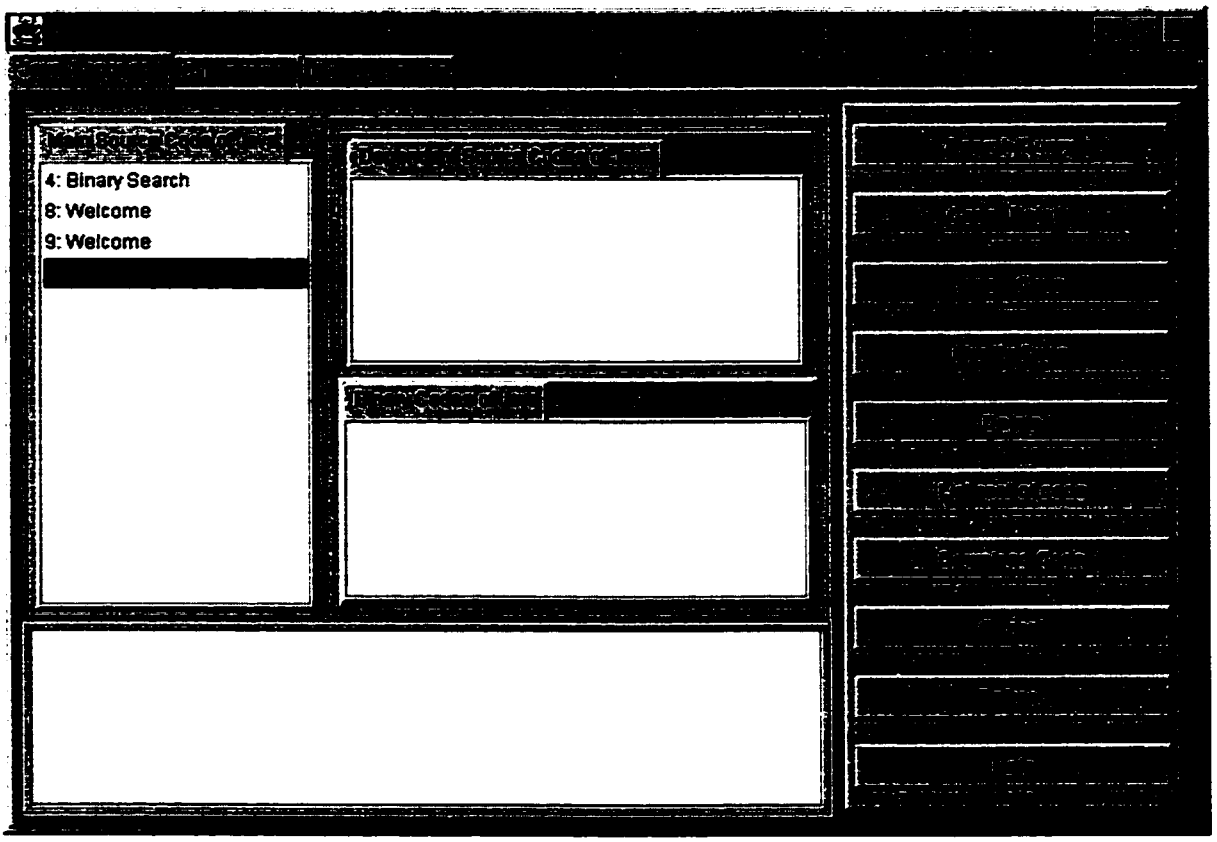


Figure 12: Code Provider Component User Main Operation Interface

Figure 12 is the code provider user main operation interface. It is the main interface of code provider component, users can do almost all operations such as query/search code, insert code, modify code, delete code, list rest of code, download code and so on here, it includes:

- **Three Tabbed Panes** are Java language tabbed pane, C language tabbed pane and C++ language tabbed pane, users can switch among the three panes according to their needs. For example, if you want to insert C++ code, you should choose the C++ language tabbed pane. Each tabbed pane also consists of a few sub-panes such as main source code pane, dependent source codes pane, binary codes pane, code display pane and button group

pane.

- **Main source code pane** is used to display the all same language type's main codes stored in the remote centralized database by names.
- **Dependent source codes pane** is used to display the dependent source codes that are corresponding to main source code displayed on the main source code pane when you click the <List rest of code> button on the right button group pane.
- **Binary codes pane** is used to display the binary codes which are corresponding to main source code displayed on the main source code pane when you click the <List rest of code> button on the right button group pane.
- **Code display pane** is used to display the content of corresponding source code when you download it.
- **Button group pane** contains ten functional buttons which are <Search/Query>, <Code Details>, <Insert Code>, <Modify Code>, <Delete Code>, <List rest of code>, <Download Code>, <Cancel> and <Refresh>, <Help>. <Cancel> button is used to cancel this operation, <Refresh> button is used to refresh the display of the main code names on the main code pane, click <Help> button to get help how to use this interface.

The following explains the functionalities of other seven buttons by examples.

7) Code Provider Component User Search/Query Interface:

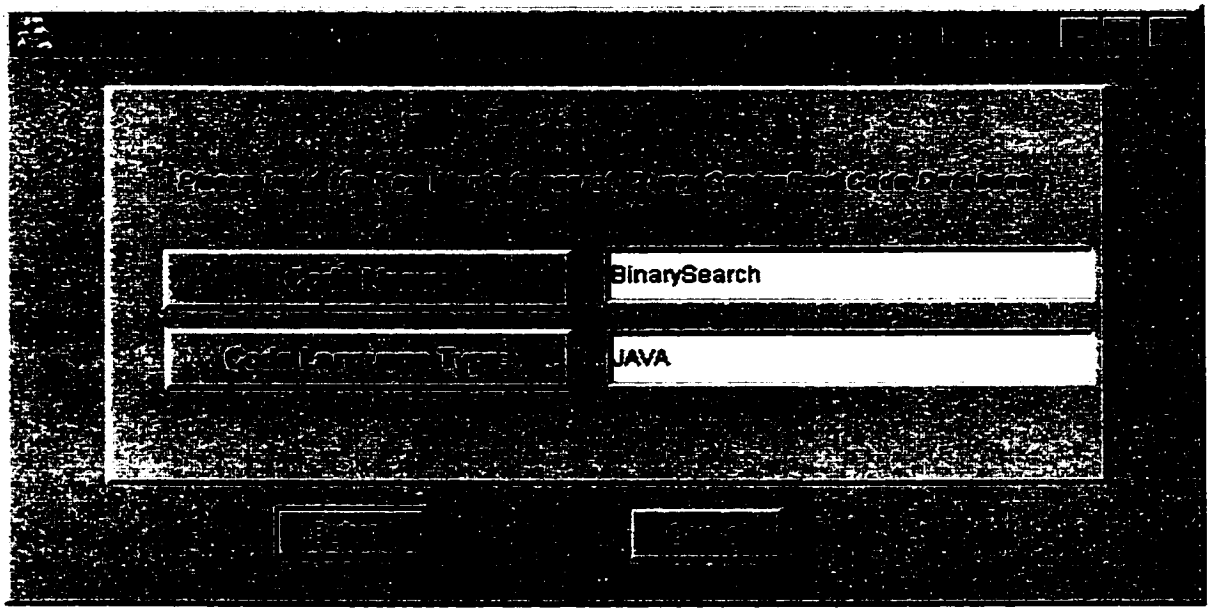


Figure 13: Code Provider Component User Search/Query Interface

Figure 13 above is code provider component user search/query interface. This interface is popped up when you click the <Search/Query> button of **figure 12**. If the users want to know the existing codes in the remote centralized code database before they want to insert their own new codes or want to find their own codes which have been inserted before, they can use this interface to find the codes by offering keywords such as code name and code type. The result will be display on the corresponding main source pane of **figure 12** after click the <Submit> button, click <Cancel> to cancel this operation.

8) Code Provider Component User Insert Code Interface:

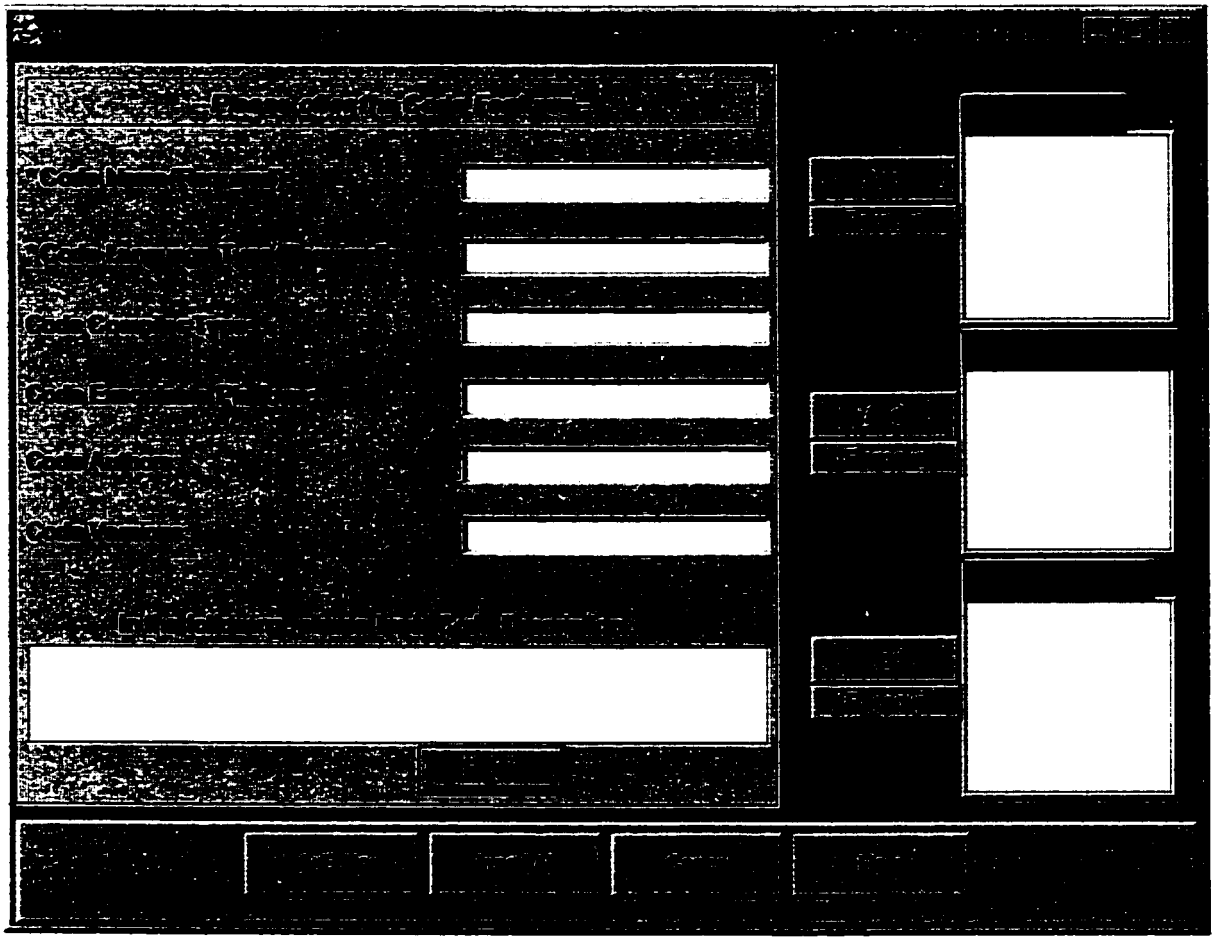


Figure 14: Code Provider Component User Insert Code Interface

Figure 14 above is code provider component user insert code interface. It is the key interface of main operation interface. The users can use this part to insert their own source codes and corresponding binary codes into the remote centralized database. It consists of three main panes that are code feature pane, add/remove code pane and button group pane.

- **Code feature pane** is used by users to specify their own code's features (code attributes) before inserting into the remote centralized database.
- **Add/Remove code pane** is used to add/remove the main code file,

dependent code files and binary code files into/from this pane.

- **Button group pane** has four buttons that are <My Editor>, <Insert All>, <Cancel> and <Help>.

The following explains how to use this interface in detail by examples.

9) Example of Code Provider Component User Insert Code Interface 1:

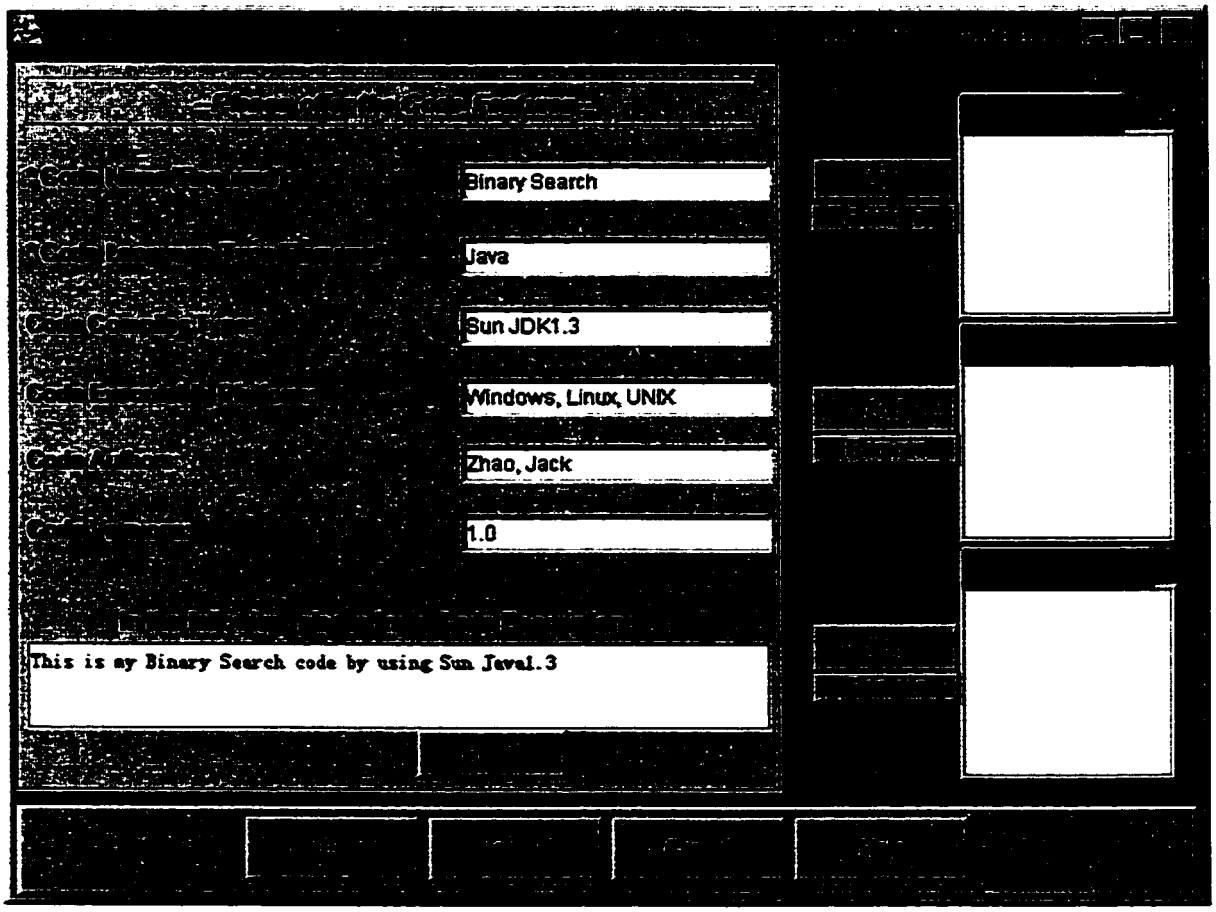


Fig. 15:Example of Code Provider Component User Insert Code Interface 1

Figure 15 above is an example of code provider component user insert code interface 1. It is the step 1 when you want to insert your own codes, in this step, you should specify your own code's features (code attributes) before inserting them into the remote centralized database on the code feature pane,

<Reset> button is used to re-specify your code features.

10) Example of Code Provider Component User Insert Code Interface 2:

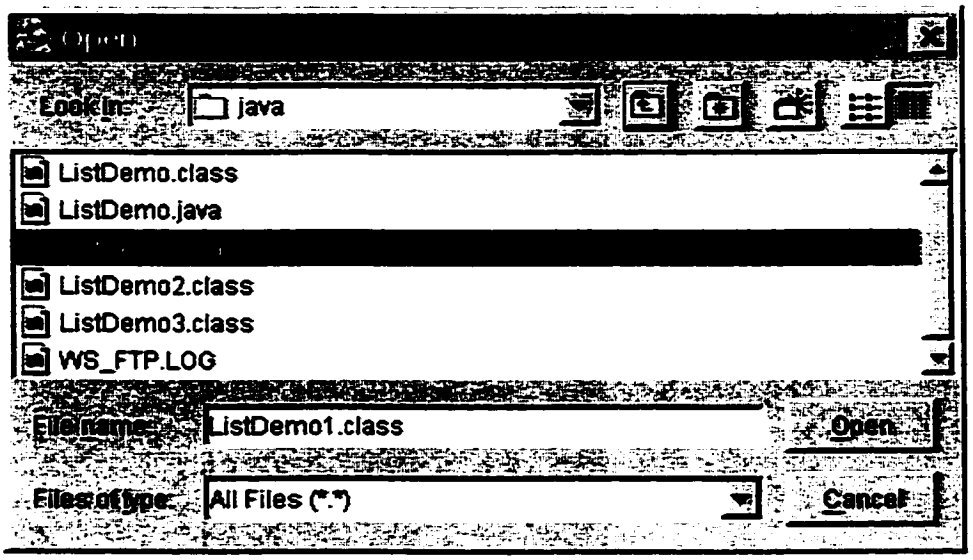


Fig.16: Example of Code Provider Component User Insert Code Interface 2

Figure 16 above is an example of code provider component user insert code interface 2. It is the step 2 for inserting codes, after you specify your own code's feature in above step 1. Click <Add> button on code pane of **figure 15** will invoke the **figure 16** interface which is the file chooser interface, you click <Add> button on the code pane of **figure 15** to invoke **figure 16** file chooser interface repeatedly to add your own all code files such as main code file, dependent code files and binary code files into the code pane. You also can use <Remove> button of **figure 15** to remove selected file on the code pane.

11) Example of Code Provider Component User Insert Code Interface 3:

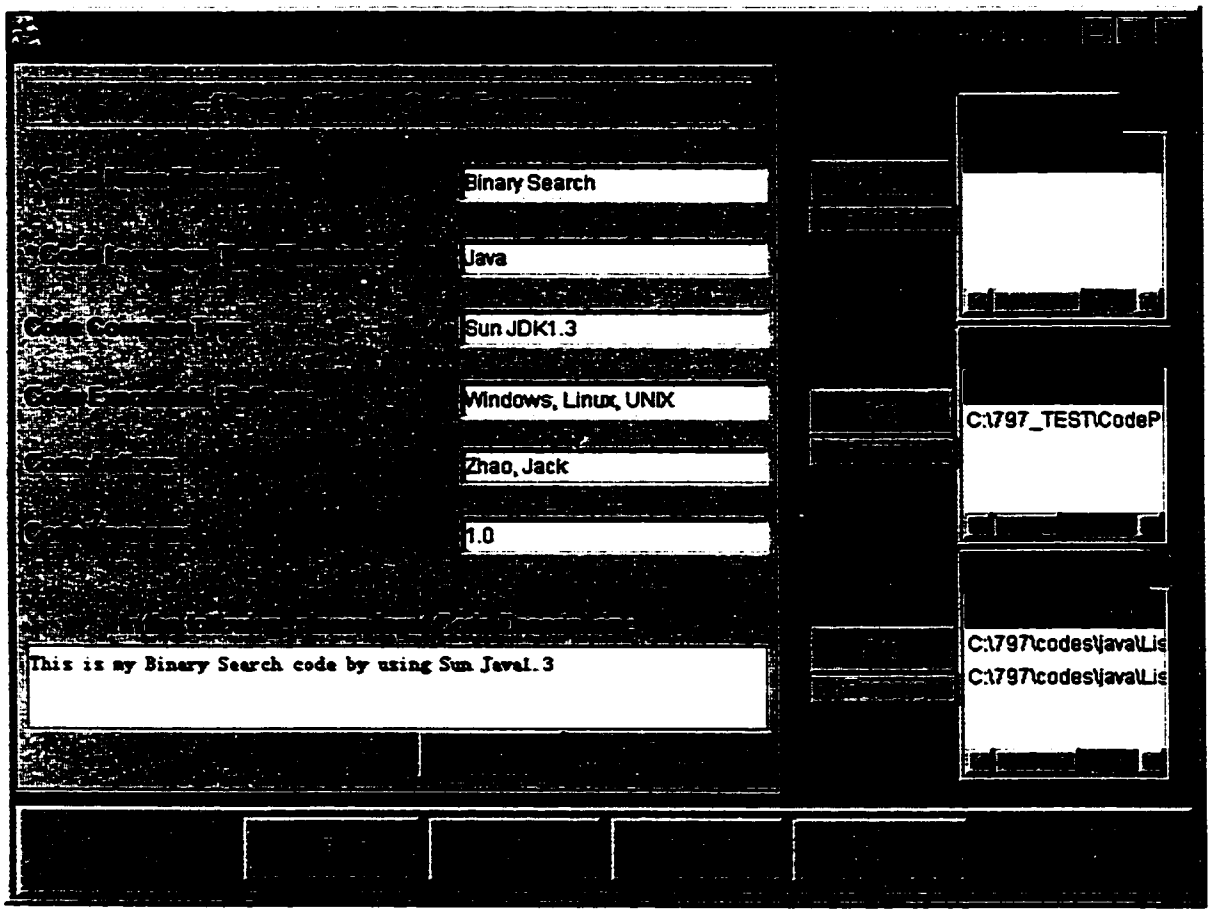
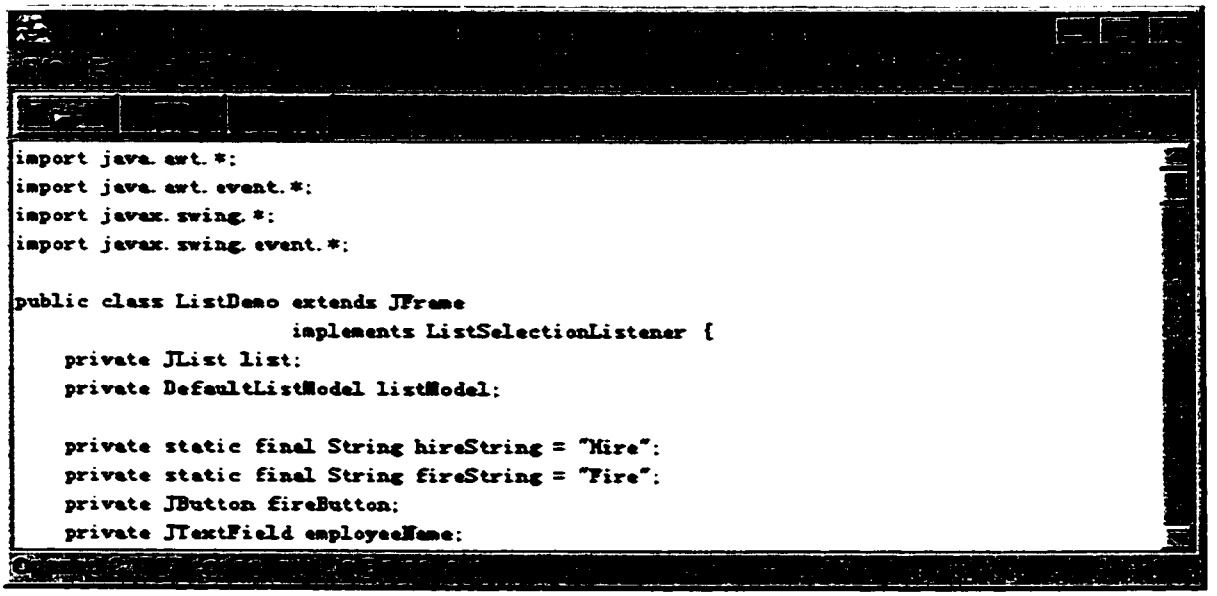


Fig.17: Example of Code Provider Component User Insert Code Interface 3

After finishing step 1 and 2, the code insert interface should look like **figure 17**, and then you can click <Insert All> button to insert all your code information and corresponding files into the remote centralized database. You also can click <My Editor> button of **figure 15** to invoke **figure 18** interface that is an editor coded by Java. You can also use it to edit, modify and browse your own code files before inserting them into the centralized database.

12) Example of Code Provider Component User Insert Code Interface 4:



```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.event.*;

public class ListDemo extends JFrame
                      implements ListSelectionListener {
    private JList list;
    private DefaultListModel listModel;

    private static final String hireString = "Hire";
    private static final String fireString = "Fire";
    private JButton fireButton;
    private JTextField employeeName;
```

Fig. 18: Example of Code Provider Component User Insert Code Interface 4

13) Code Provider Component User View Code Detail Interface:

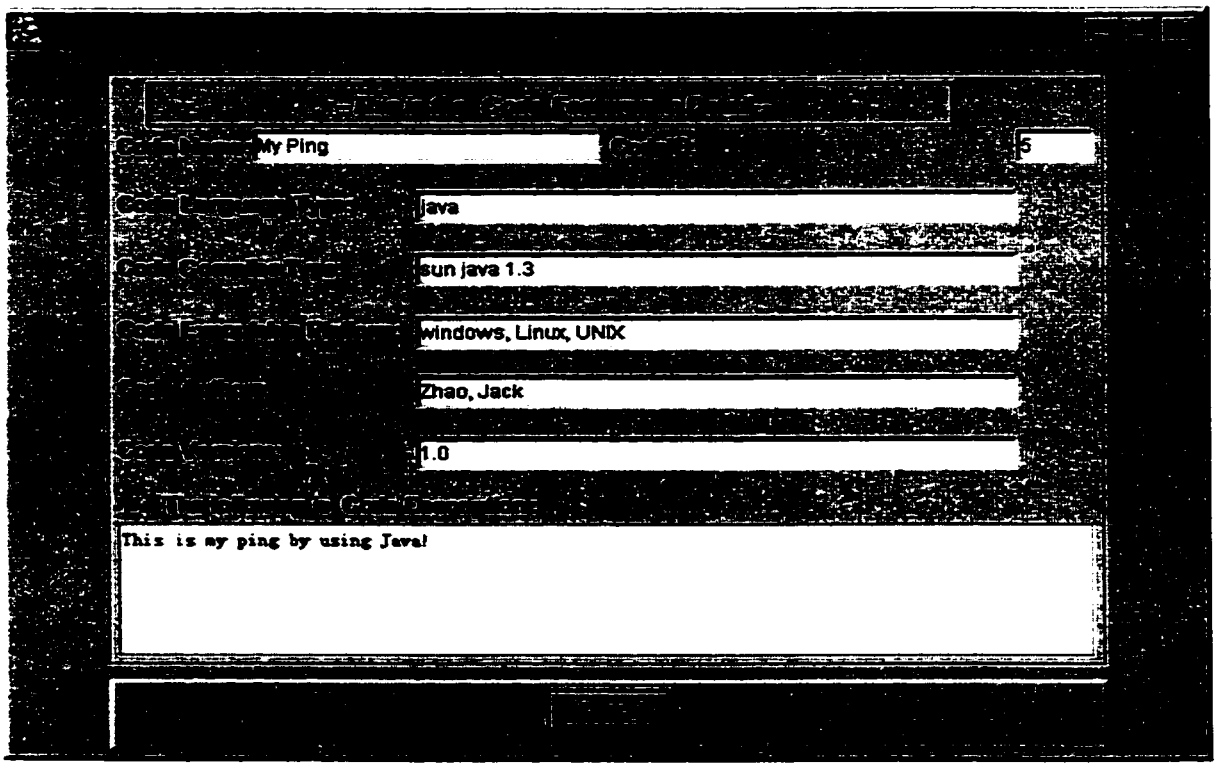


Figure 19: Code Provider Component User View Code Detail Interface

If the uses want to view the code information in detail, they should select the code name on the main source code pane and highlight it in the **figure 12** interface, and then click <Code Details> button of **figure 12** interface. **Figure 19** interface will be popped up, it is the code provider user view code detail interface. <Cancel> button is used to cancel this operation.

14) Code Provider Component User Modify Code Interface.

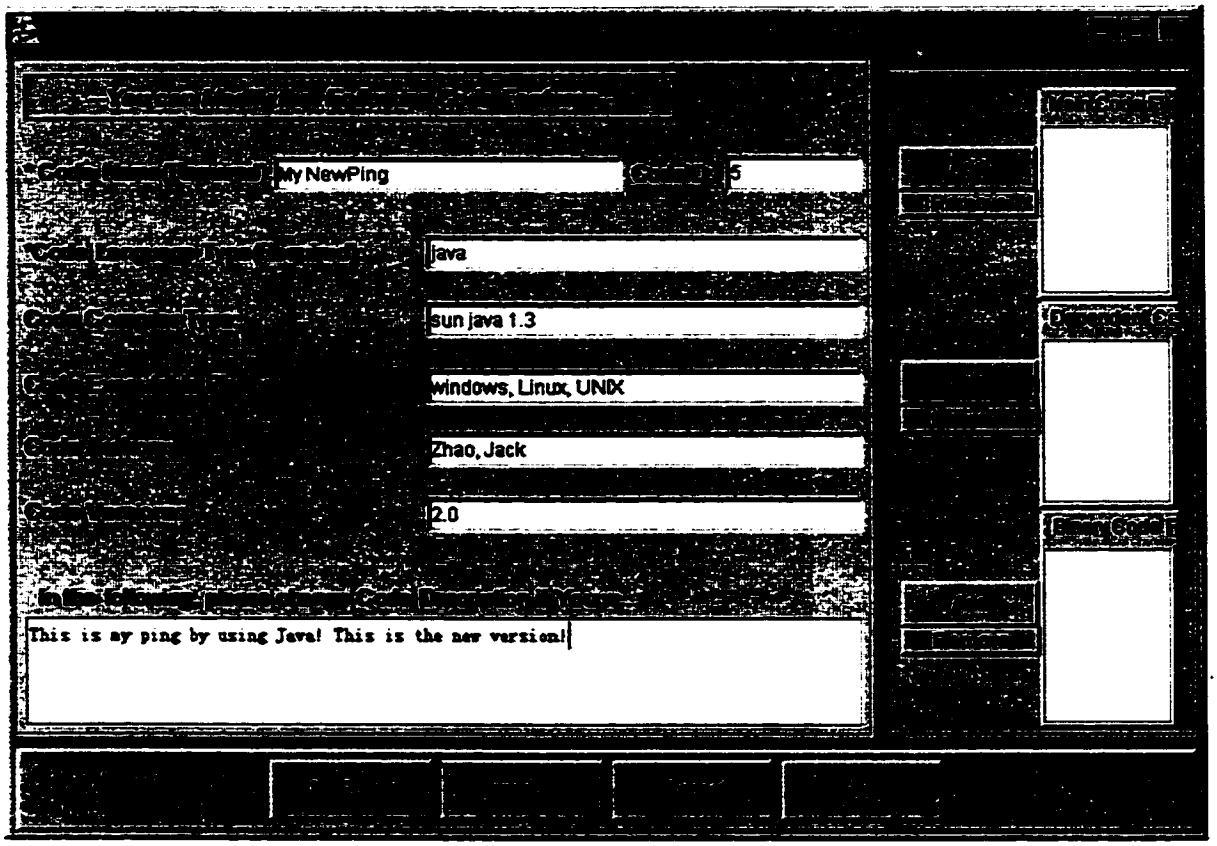


Figure 20: Code Provider Component User Modify Code Interface

If the uses want to modify their code features, source codes and corresponding binary codes which have been inserted into the remote centralized database before, they should select the code name which is wanted to modify on the main source code pane and highlight it in the **figure 12** interface, and then click <Modify Code> button of **figure 12** interface. **Figure 20** above interface will be popped up, it is code provider component user modify code interface. It is very similar to the code provider component user insert code interface **(figure 14)**. After modifying the code features and add modified source code file, corresponding dependent code files and binary code files, then click <Insert All> button to store the new code information into the remote centralized database. The rest of operations for this interface are very similar

to the code provider component user insert code interface(**figure 14**), please refer to it.

15) Code Provider Component User Delete Code Interface:

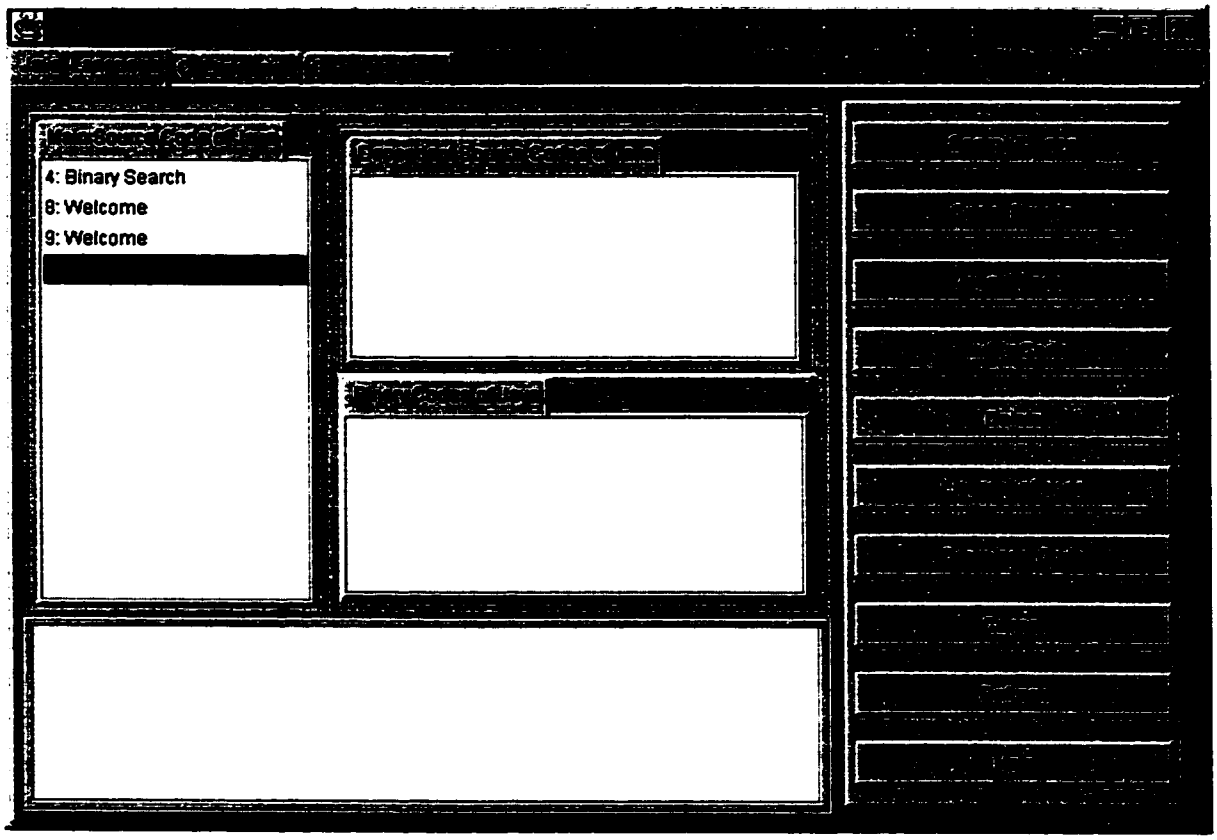


Figure 21: Code Provider Component User Delete Code Interface

Figure 21 above is code provider component user delete code interface, if the user has the deletion privilege, he/she can use this interface to delete the source codes and corresponding binary codes from remote centralized database. For example, the user should select code name which will be deleted on the main source code pane and highlight it in the **figure 12** interface, and then click <Delete Code> button to delete it, because deletion is very dangerous, after <Delete Code> is clicked, the system will pop up **figure 22** dialog box to warn the user if he/she want to delete this code truly.

16) Code Provider Component User Delete Code Dialog Box:

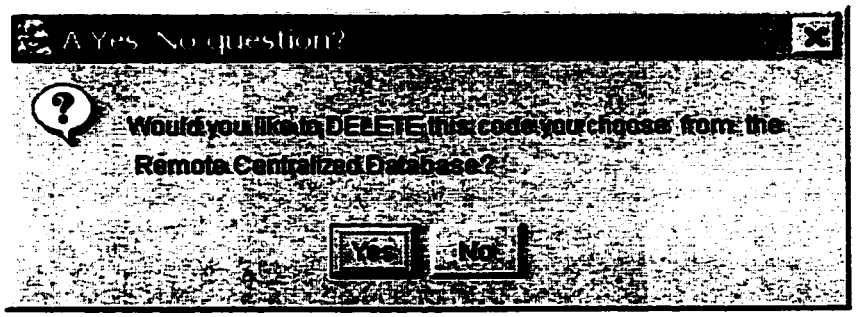


Figure 22:Code Provider Component User Delete Code Dialog Box

17) Code Provider Component User List Rest of Code Interface:

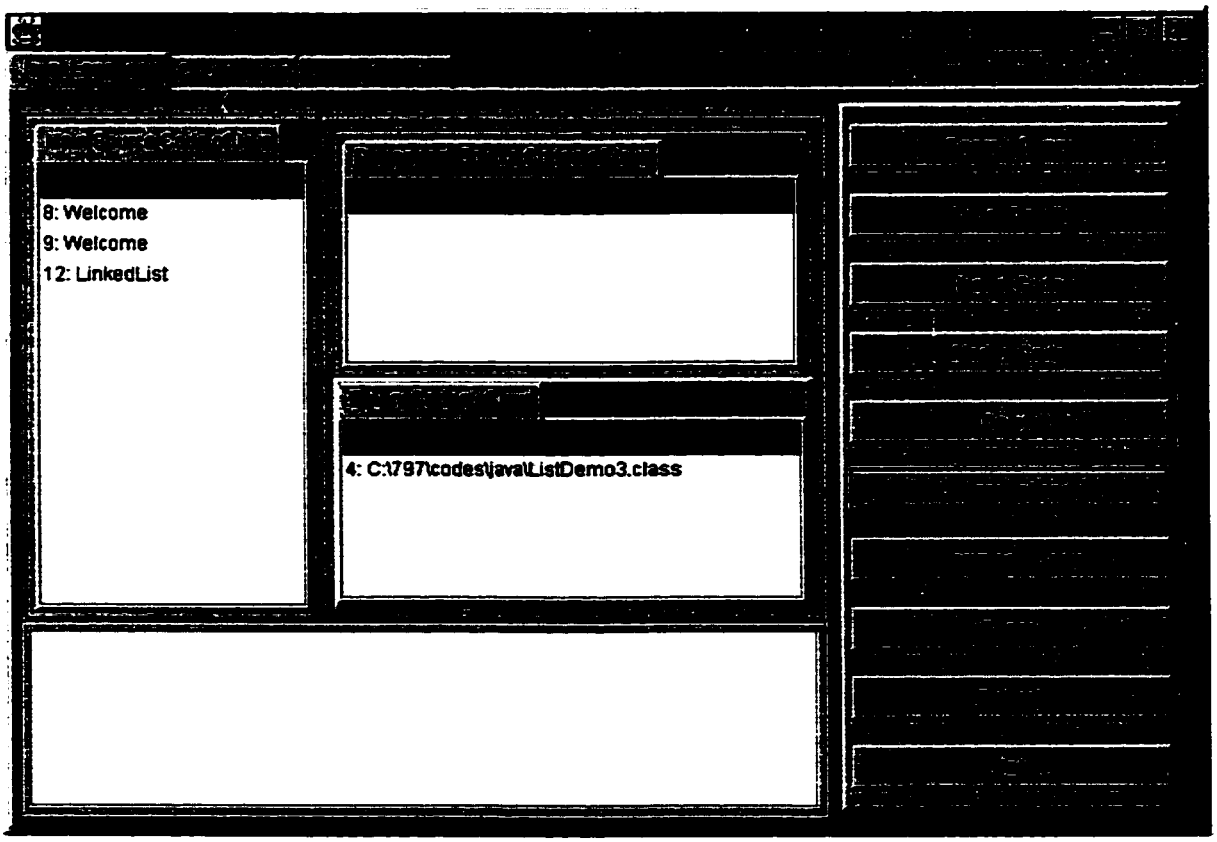


Figure 23: Code Provider Component User List Rest of Code Interface

Figure 23 above is code provider component user list rest of code interface. This interface is used to display the rest of code parts such as dependent code

files and binary code files. For example, the user should select code name for which the rest of code files will be displayed on the main source code pane and highlight it, and then click <List rest of code> button to display them.

18) Code Provider Component User Download Code Interface:

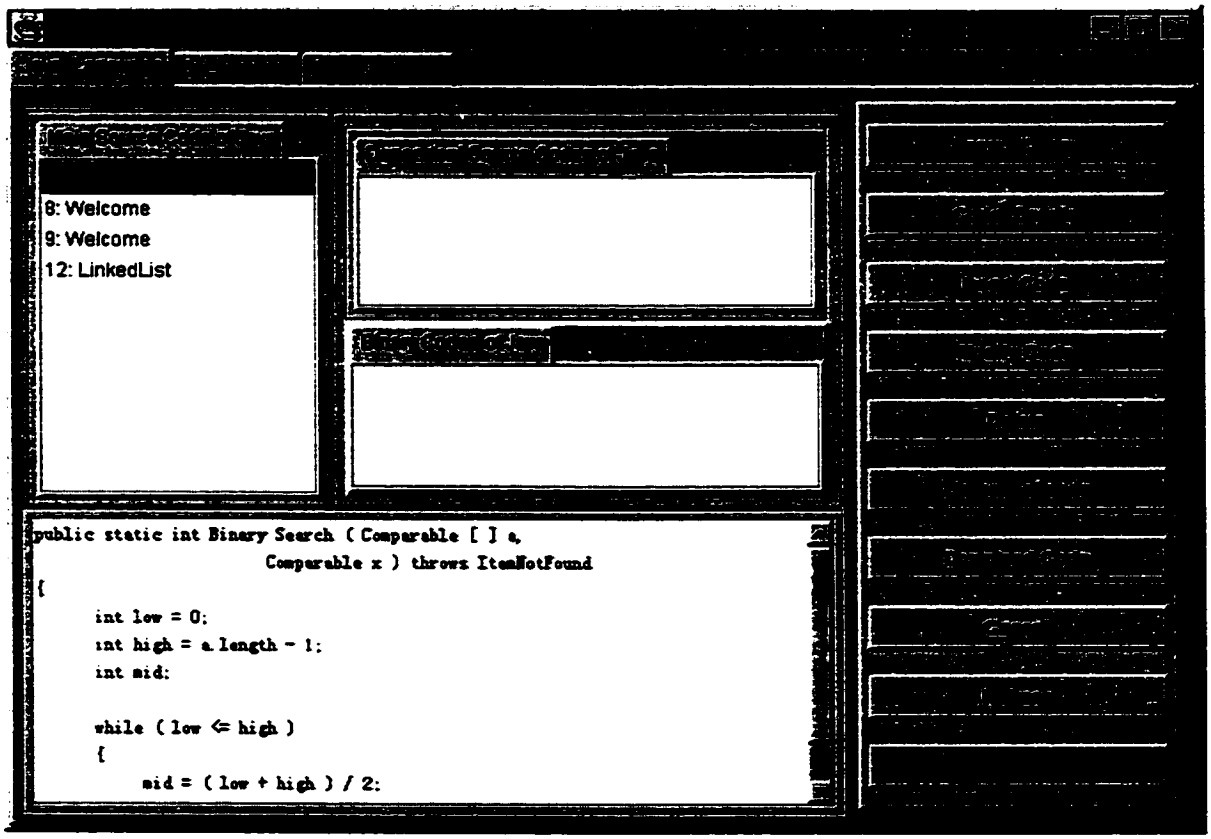


Figure 24: Code Provider Component User Download Code Interface

Figure 24 above is code provider component user download code interface. The users use this interface to download the actual main codes, dependent codes and binary codes which they want from remote centralized database into their own computer. For example, the user select code name which is wanted to download on the main source code pane and highlight it, and then click <Down Code> button, the **figure 25** code save interface will be popped up and ask you to select the file saving location repeatedly until all main code file, corresponding dependent code files and binary code files are downloaded and

saved into your own computer.

19) Code Provider Component User Download Code Save Interface:

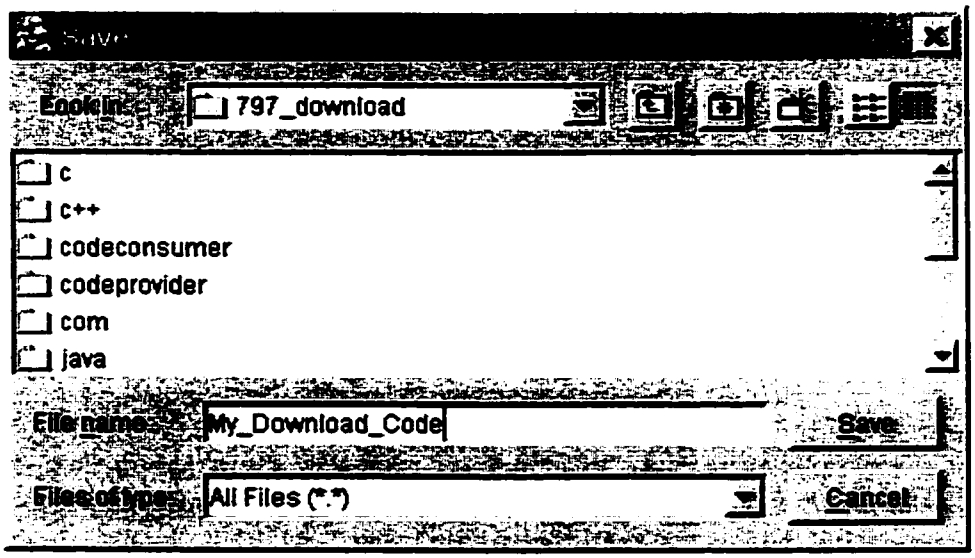


Figure 25: Code Provider Component User Download Code Save Interface

4.7.5: Code Consumer Component's Interfaces:

In the above section, we have discussed how to use the interfaces of code provider component. In the following section, we will discuss how to use interface of code consumer component, the users of code consumer component only need to query/search, browse the codes in the remote centralized database, and download the useful codes from the database, they have no privileges to insert, modify, delete the codes of centralized database.

The interfaces of code consumer component are very similar to the interfaces of code provider component, please refer to the interfaces of code provider component to use it.

1) Code Consumer Component Welcome Interface:

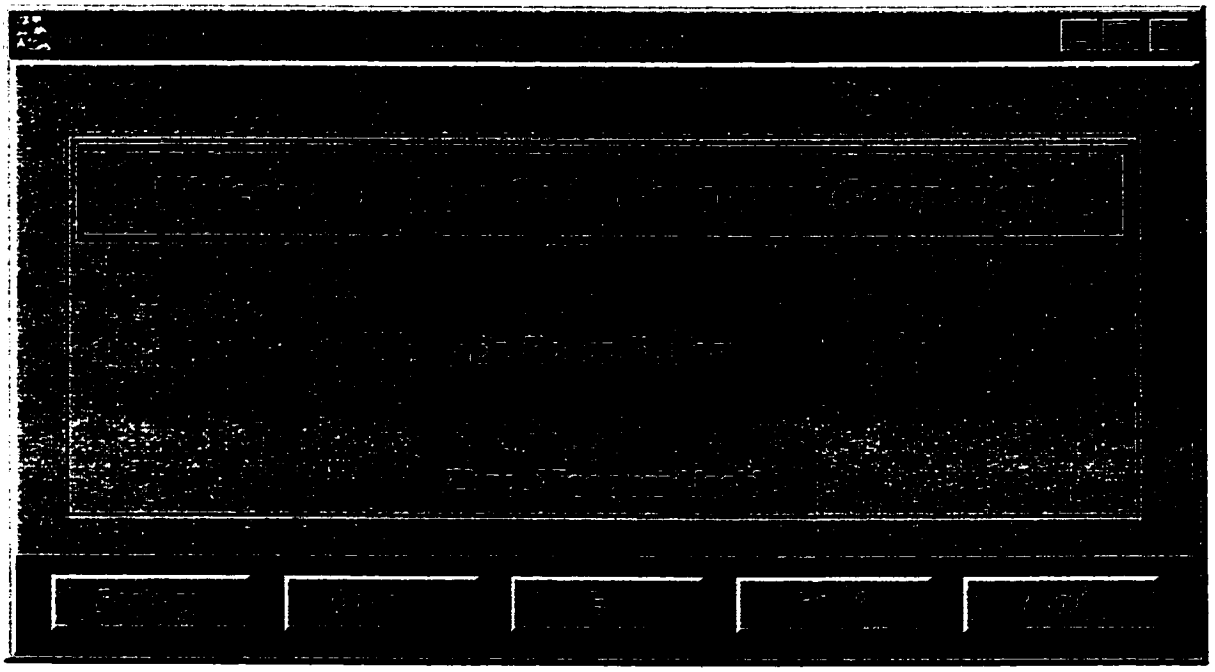


Figure 26: Code Consumer Component Welcome Interface

Figure 26 above shows the first welcome interface after you run the code consumer component. This interface has five buttons: You can click <Register> button to register yourself before you want to user this code consumer component; if you already have registered and had valid userID and password, please click <Come in> button to enter the code consumer component; press <Exit> button to exit this component; press <Help> button to get help how to use this interface; press <About> button to get system information.

2) Code Consumer Component User Operation Option Interface:

The **figure 27** interface allows user to do either "Enter code consumer component" or "Enter user account management component" by clicking corresponding button. If you want to change your register information on line, you should click the button behind "Enter user account management

component"; If you want to search/query, browse the codes in the remote centralized database and download useful codes, you should click the button behind "Enter code consumer component".

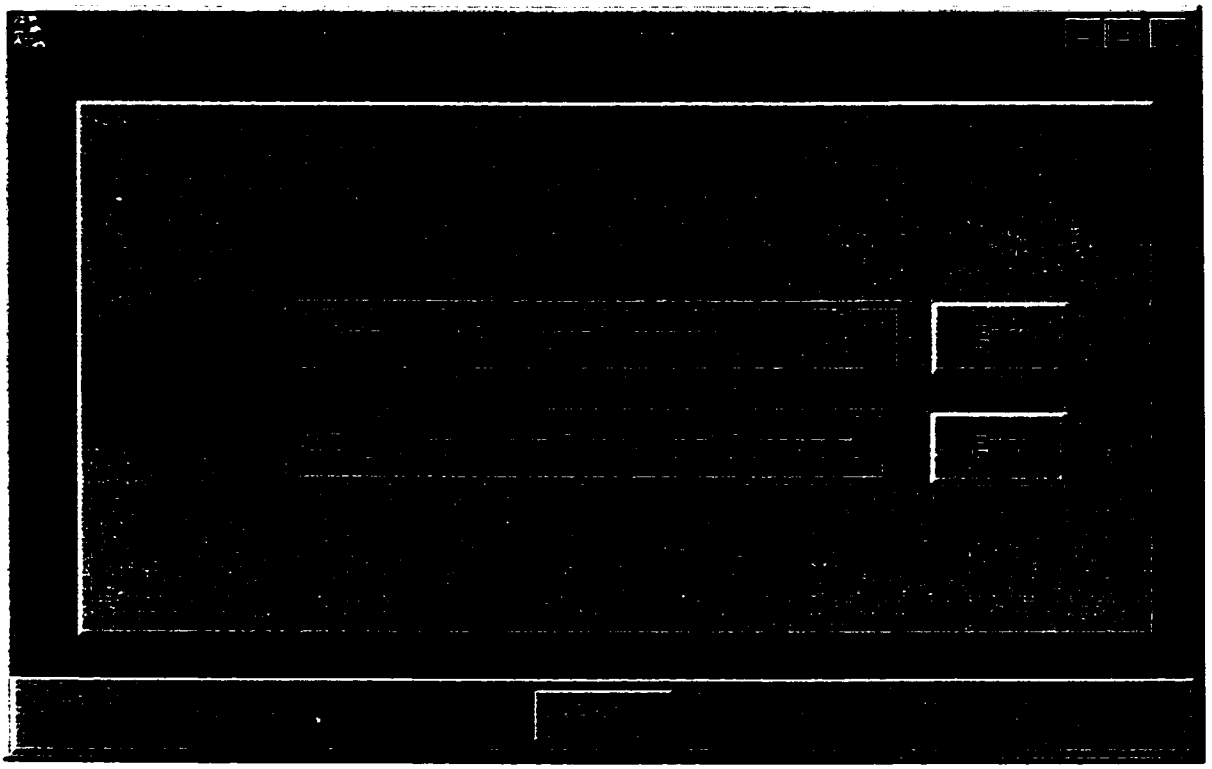


Figure 27: Code Consumer Component User Operation Option Interface

3) Code Consumer Component User Main Operation Interface:

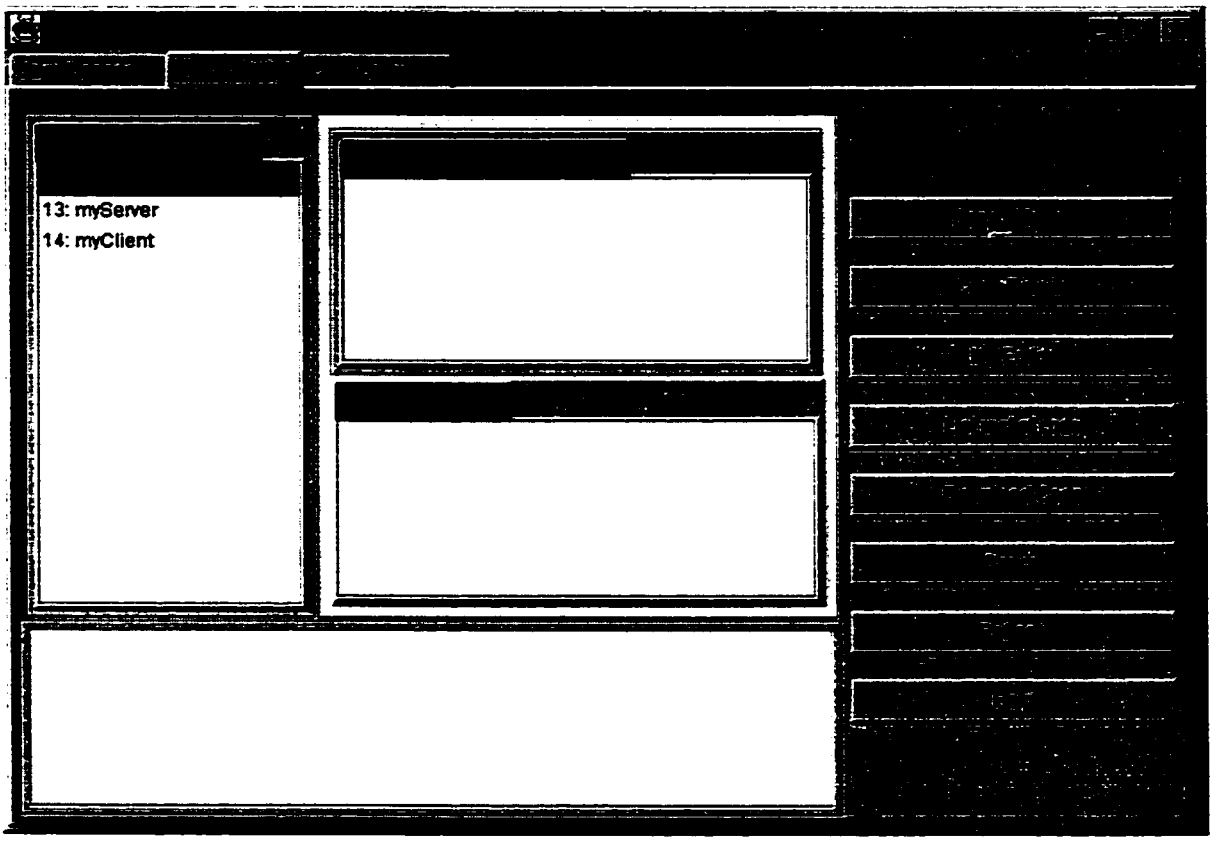


Figure 28: Code Consumer Component User Main Operation Interface

Figure 28 above is the code consumer component user main operation interface, users can do operations such as query/search code, view code detail, list rest of code, and download code etc. here. It includes three tabbed panes: Java language tabbed pane, C language tabbed pane and C++ language tabbed pane. Each tabbed pane also consists of a few sub-panes such as main source code pane, dependent source codes pane, binary codes pane, code display pane and button group pane. Button group pane contains eight functional buttons, which are <Search/Query>, <Code Details>, <My Editor>, <List rest of code>, <Download Code>, <Cancel> and <Refresh>, <Help>. Click <My Editor> to invoke a Java editor, for the rest of seven buttons' functionalities, please refer to the usages of code provider component's interfaces above.

Chapter 5: System Testing, Thesis Conclusion and Future Work

5.1: System Testing:

Our system's code provider component, code consumer component and centralized database components have been tested on the following environments successfully.

5.1.1 Testing on Personal Computer Environment

For system test, we installed three operating systems on my personal computer that are Microsoft Windows98, Microsoft 2000 and Red Hat Linux7.0; we also installed oracle8i object-relational database for each of the three platforms.

(1) Windows98 platform: we installed oracle8i database server for Windows98 as centralized code services database, and Java Runtime Environment (JRE) for Windows98, centralized database server's TCP/IP networking configuration is: the IP address is:127.0.0.1(domain name: localhost), listening port is:1521; the networking services ID(SID) is ORCL. We use code provider component and code consumer component to communicate with the centralized database to test the system functionalities, all system functionalities worked well.

(2) Windows2000 platform: we installed oracle8i database server for Windows2000 as centralized code services database server, and Java Runtime Environment(JRE) for Windows2000. The centralized database server's TCP/IP networking configuration is same as Windows98.

(3) Red Hat Linux7.0 platform: we installed oracle8i database server for Red Hat Linux7.0 as centralized code services database server, and Java Runtime Environment(JRE) for Red Hat Linux7.0. The centralized database server's TCP/IP networking configuration is also same as Windows98.

For the above both Windows2000 platform and Red Hat Linux7.0 platform, we use code provider component and code consumer component to communicate with the corresponding centralized database, all system functionalities also worked well.

5.1.2: Testing on Distributed Networking Environment

After testing on single personal computer environment successfully, in order to test our system on a large scale's networking environment, we established the following networking environment for testing:

- **Establish web page:** We establish a temporary's web page on our Computer

Science department's web site where the users can download the code provider component and code consumer component into their own personal computer. The URL is http://davinci.newcs.uwindsor.ca/~zhaoc/.

- **Establish centralized code services database:** We established centralized database on our computer science department's database server which is Oracle8i database running on Sun Solaris Unix5.7, its TCP/IP networking configuration is: the IP address is:137.207.76.4(domain name is: goedel.newcs.uwindsor.ca); listening port is:1521; and, the networking services ID(SID) is cs01.

For testing, we download the code provider component and code consumer

component from above web page by using IE or Netscape browsers into **SGI Server (IP:137.207.16.4), Davinci Server (IP:137.207.76.3), HPC Server(IP: 137.207.154.3).** and then use the two components to communicate with the remote centralized database(our dept's database server, IP:137.207.76.4). All system functionalities worked well.

We also downloaded the two components into my computer and my friends' computers at home, all the Internet connections are connected by 56K Modem, Cogeco's Cable Modem or Bell DSL, the two components also worked well.

All these tests are under large scale distributed networking environments, our system is proved to run successfully.

5.1.3: Testing the Long Transaction

Because insertion and modification operation of code services are used in our system frequently, all these operations are long transaction. It is necessary to test the long transaction.

In order to test the data integrity of main code, dependent code and binary code during the transaction, I use the following steps to test the long transaction. Please note all change operations for main code, dependent code and binary code are treated as one long transaction. This transaction is either committed or roll backed.

1) Select the codes: Use "User Modify Code Interface" to select new version of large main code file, dependent code files and binary code files.

2) Make network connection failure: First click "Insert All" button to begin the long modification transaction, then disconnect the network connection

intentionally by myself during the transaction. Since all these files are large, it takes some time to transfer, so, we can guarantee that the network fails during the long modification transaction.

3) Check if the codes are modified: Connect the network again and check if these main code, dependent code and binary code are changed. The answer is "NO".

4) Result: Because the network fails during the long modification transaction. This long transaction is roll backed, not is committed, no any change to the centralized database. This is to prove that our system supports the long transaction. Actually, our system is based on the object-relational database and JDBC, they both support data transaction.

5.2: Thesis Conclusion

In this thesis, we focused on how to store the code services and find which system architecture is suitable for code services retrieval system.

Our solution is to integrate the object-relational database technology, Java JINI concept and JDBC technologies together to establish: A Centralized Object-Relational Database Based Code Service Retrieval System Tool for Software Reuse. In our system, we created our own user-defined data type including CLOB, BLOB to store source codes and corresponding binary codes. Centralized object-relational database is accessed by remote users, the centralized database is flexible, which can be put in anywhere. We also used new pure Java client-side JDBC to access our own defined data types including CLOB and BLOB in our system. Our system is access-effective. It also supports collaborative programming by multi-users concurrently.

5.3: Future Work

- Improve the system security. At present, the system provides basic user authentication. But, the code access is not secure. Additionally, one should add encryption for the codes to support secure transmission.
- Using distributed database servers to share the load of centralized database. This may involve amalgamating previous work on multi-server approaches by Xiaohong Yu (M.Sc. Thesis 2001).
- Centralized database has bottleneck problem, if a large amount of users access our system at same time, the single centralized database may not handle so many connections efficiently. This is bottleneck problem.

REFERENCES:

[BAKER97]: Sean Baker, "CORBA Distributed Objects Using Orbix", Addison-Wesley, 1997.

[BOOCH99]: Grady Booch, James Rumbaugh, " The Unified Modeling Language User Guide", Addison-Wesley, 1999.

[BOUG00]: Athman Bouguettaya,Boualem Benatallah, "Using Java and CORBA for Implementing Internet Databases", Queensland University of Technology - GPO Box 2334 Brisbane, QLD 4001 Australia.

[CHAS00]: Julien Chastang, "Java and Databases".

[CORBA97]: "The Common Object Request Broker: Architecture and Specification", Version 2.1,Object Management Group, August 1997.

[COSS97]: "CORBA Services: Common Object Services Specification " Object Management Group, July 1997.

[DOWN98]: T. B. Downing, "Java RMI: Remote Method Invocation", IDG Books Worldwide, 1998.

[DAVID00]: David A. James, "A Common Interface to Relational Databases from R and S", Bell Labs, Lucent Technologies,March 2000.

[DAY00]: Bill Day, " Jini Connection: Technology Architecture Overview", Technology Evangelist, Sun Microsystems.

[ENOS99]: Baat Enosh, "Java and Databases", May 3, 1999.

[FRAK94]: W. B. Frakes and S. Isoda," Success Factors of Systematic Reuse", IEEE Software, Vol.11, No.5, Sept. 1994, pp.15—19, Introduction to the special issue on "Systematic Resue".

[HELM97]: G. Helmayer, G. Kappel, and S. Reich, "Connecting Databases on the Web: A Taxonomy of Gateways", Eighth Int. DEXA Conference, Sept. 1997.

[IBM00]: "Object-Relational DB2, White Paper, Data Management Solution", IBM Company, 2000.

[JDBC01]: http://www.javasoft.com/products/jdbc/jdbc.drivers.html

[KELLER01]: Wolfgang Keller, "Mapping Objects to Tables A Pattern Language",http://ourworld.compuserve.com/homepages/WofgangWKeller/, http://www.sdm.de/g/arcus/.

[KRUE92]: W. Krueger, " Software Reuse", ACM, June 1992.

[LAMB00]: A. Lambrinidis and N. Rousopoulos, "Generating dynamic content at database-backed web server, cgi-bin vs mod_perl", Sigmod Record, March 2000.

[LEO99]: Zhengyu Leo Wu, "Software Reuse and World Wide Web", University of Windsor,1999.

[LIM94]: M.C. Lim, " Effects of Reuse on Quality, Productivity and Economics", IEEE Software, Vol.11, No.5, Sept. 1994, pp.23—30.

[LIND00]: Peter van der Linden "Just Java and beyond", 2000.

[MACHA96]:Salah-Eddine Machani, "Events in an Active Object-Relational Database System", Studies in Science and Technology, Master's Thesis, September 1996.

[MSQL01]: http://www.edm2.com/0607/msql3.html

[NIEM00]: Patrick Niemeyer, Joshua Peck, "Exploring Java".

[ORACLE00]: "Oracle8i Application Developer's Guide - Large Objects (LOBs)", Release 2 (8.1.6) December 2000.

[ORACLE01]: "Oracle8i JDBC Developer's Guide and Reference", Release 3 (8.1.7), July 2001.

[ORACLE99]: "Oracle8i Application Developer's Guide - Object-Relational Features", Release 2 (8.1.6), December 1999.

[ORFA98]: R. Orfali, D. Harkley, "Client Server Programming with Java and CORBA", Second Edition. Whiley Publishing, 1998.

[PAPA00]: Stavros Papastavrou, Panos Chrysanthis,George Samaras,Evaggelia Pitoura, "An Evaluation of the Java-based Approaches to Web Database Access", Dept. of Computer Science, University of Pittsburgh.

[QIANG99]: June Xuejun Qing, "Organizing Imperative Programs for Execution-Based Retrieval for Reuse".

[QUOIN00]: "Distributed Computing Overview", QUOIN, Cambridge, Massachusetts 02138.

[RAMAK00]: Raghu Ramakrishnan, "Database Management Systems", University of Wisconsin, Madison, WI, USA.

[RMI97] "Java Remote Method Invocation", Sun Microsystems, Inc., December 1997.

[SIEGEL96]: Jon Siegel, "CORBA Fundamentals and Programming", John Wiley & Sons, Inc., 1996.

[STON96]: M. Stonebraker, "Object-Relational DBMSs: The Next Great Wave",Morgan Kaufmann Publishers, 1996.

[SUBRA00]: Subrahmanyam Allamaraju, Karl Avedal, "Professional Java Server Programming J2EE Edition", Wrox Press Ltd. 2000.

[SUN00]: Sun Company, "Jini Architectural Overview", Technical White Paper, 2000.

[UNGLA00]: F. Unglauben, W. Hillen, M. Murdfield, Medizinische Informatik "Evaluation of Two- and Three-Tier Database Connections for a Java Based Medical Image Viewer", Biomedizinische Technik Ginsterweg, 2000.

[VARAD99]: Dilip Varadarajan, "A Three-tier Client/Server Architecture for Scientific Data Retrieval", Oregon State University, 1999.

[XHUMA97]: Florian Xhumari, "Java Universal Binding: Storing Java Objects in Relational and Object-Oriented Databases", August,1997, O2 Technology.

[ZHANG00]: Michael Hui Zhang "Design and Construction of a Library-Based Software Reuse Model to Support Distributed and Grid Computer", University of Windsor, 2000.

[ZHONG00]: Sheng Zhong, "Software Library for Reuse-Oriented Program Development", University of Windsor, 2000.

APPENDIX A:

We can define our own user-defined data type that contains CLOB and BLOB to store the source codes and binary codes, and then define a table by using our own user-defined data type. We need two steps, firstly, we create the user-defined data type, we define this data type as an object type that contains the LOB attributes, second, we can proceed to create a table that makes use of that user-defined data type.

The following is our own user-defined data types and corresponding tables, we define these tables using corresponding our own defined data types. We put their definitions into a *.SQL file, this file is used to establish the tables.

```
/* 1: DROP ALL TABLES */

DROP TABLE CONSUMER_USER_TABLE    CASCADE CONSTRAINTS;
DROP TABLE PROVIDER_USER_TABLE    CASCADE CONSTRAINTS;
DROP TABLE MAIN_CODE_TABLE        CASCADE CONSTRAINTS;
DROP TABLE DEPENDENT_CODE_TABLE CASCADE CONSTRAINTS;
DROP TABLE BINARY_CODE_TABLE      CASCADE CONSTRAINTS;

/* 2: DROP ALL TYPES*/

DROP TYPE  CONSUMER_USER_TABLE_TYPE;
DROP TYPE  PROVIDER_USER_TABLE_TYPE;
DROP TYPE  MAIN_CODE_TABLE_TYPE;
DROP TYPE  DEPENDENT_CODE_TABLE_TYPE;
DROP TYPE  BINARY_CODE_TABLE_TYPE;
```

```
/* 3: CREATE ALL TYPES */

    CREATE TYPE CONSUMER_USER_TABLE_TYPE AS OBJECT
    ( LASTNAME                    VARCHAR2(35),
      FIRSTNAME                   VARCHAR2(35),
      USERID                      NUMBER(4),
      PASSWORD                    VARCHAR2(6),
      EMAIL                       VARCHAR2(50),
      ADDRESS                     VARCHAR2(70),
      USERCONFIRMED               VARCHAR2(1),
      IFSHARCNET                  VARCHAR2(1),
      IFC3                        VARCHAR2(1),
      IFGUEST                     VARCHAR2(1),
      PRIVILEGE                   VARCHAR2(1)
    );
/

    CREATE TYPE   PROVIDER_USER_TABLE_TYPE AS OBJECT
    ( LASTNAME                    VARCHAR2(35),
      FIRSTNAME                   VARCHAR2(35),
      USERID                      NUMBER(4),
      PASSWORD                    NUMBER(6),
      EMAIL                       VARCHAR2(50),
      ADDRESS                     VARCHAR2(70),
      USERCONFIRMED               VARCHAR2(1),
      IFSHARCNET                  VARCHAR2(1),
      IFC3                        VARCHAR2(1),
      IFGUEST                     VARCHAR2(1),
      PRIVILEGE                   VARCHAR2(1)
    );
```

```
/
    CREATE TYPE MAIN_CODE_TABLE_TYPE AS OBJECT
    (CODEID                 NUMBER(5),
     USERID                 NUMBER(4),
     FILENAME               VARCHAR2(90),
     CODENAME               VARCHAR2(50),
     CODETYPE               VARCHAR2(20),
     COMPILERTYPE           VARCHAR2(20),
     PLATFORM               VARCHAR2(30),
     AUTHOR                 VARCHAR2(40),
     VERSION                VARCHAR2(30),
     CODEDESCRIPTION        VARCHAR2(800),
     CODE                   CLOB
    );
/

   CREATE TYPE   DEPENDENT_CODE_TABLE_TYPE AS OBJECT
    (CODEID             NUMBER(5),
     FILENAME           VARCHAR2(90),
     CODE               CLOB
    );
/

    CREATE TYPE BINARY_CODE_TABLE_TYPE AS OBJECT
    (CODEID             NUMBER(5),
     FILENAME           VARCHAR2(90),
     CODE               BLOB
    );
/
```

```
/* 4: CREATE ALL TABLES */

CREATE TABLE CONSUMER_USER_TABLE OF
CONSUMER_USER_TABLE_TYPE
  ( USERID  CONSTRAINT PK_CONSUMER PRIMARY KEY);


CREATE TABLE PROVIDER_USER_TABLE OF
PROVIDER_USER_TABLE_TYPE
  ( USERID  CONSTRAINT PK_PROVIDER PRIMARY KEY );


CREATE TABLE MAIN_CODE_TABLE OF
MAIN_CODE_TABLE_TYPE
( CODEID CONSTRAINT PK_CODE PRIMARY KEY,
USERID CONSTRAINT FK_USERID_MAIN REFERENCES
PROVIDER_USER_TABLE(USERID) );


CREATE TABLE DEPENDENT_CODE_TABLE OF
DEPENDENT_CODE_TABLE_TYPE
(CODEID CONSTRAINT FK_CODE1 REFERENCES
MAIN_CODE_TABLE(CODEID) );


CREATE TABLE BINARY_CODE_TABLE OF
BINARY_CODE_TABLE_TYPE
(CODEID CONSTRAINT FK_CODE2 REFERENCES
MAIN_CODE_TABLE(CODEID));
```

VITA AUCTORIS

Zhao, Xiaoquan(Jack) was born in XianYang, China. He got his Bachelor of Science from Xian Institute of Technology, Xian, P.R.China. He is currently a candidate for the Master's degree in Computer Science (M.Sc.) at School of Computer Science in University of Windsor, Windsor, Canada.